Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Desenvolvimento de interoperabilidade entre linguagens para o *solver* Algencan 4.0

Autor: Fellipe dos Santos Araujo

Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Brasília, DF

2023

Fellipe dos Santos Araujo

Desenvolvimento de interoperabilidade entre linguagens para o *solver* Algencan 4.0

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Brasília, DF

2023

Fellipe dos Santos Araujo
Desenvolvimento de interoperabilidade entre linguagens para o *solver* Algencan 4.0/ Fellipe dos Santos Araujo. – Brasília, DF, 2023-
98 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. John Lenon Cardoso Gardenghi

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2023.

Palavras-chave: algencan. interoperabilidade. otimização. I. Prof. Dr. John Lenon Cardoso Gardenghi. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Desenvolvimento de interoperabilidade entre linguagens para o *solver* Algencan 4.0

CDU 02:141:005.6

Fellipe dos Santos Araujo

Desenvolvimento de interoperabilidade entre linguagens para o *solver* Algencan 4.0

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 15 de fevereiro de 2023:

Prof. Dr. John Lenon Cardoso Gardenghi
Orientador

Prof. Dr. Bruno César Ribas
Convidado 1

Prof. Dr. Glauco Vitor Pedrosa
Convidado 2

Brasília, DF
2023

Resumo

Problemas de otimização são problemas que não possuem algoritmos de tempo polinomial para sua resolução no caso geral. Por esse motivo, utilizam-se os *solvers*, que são *software* que utilizam algoritmos que convergem a uma solução aproximada do problema de otimização de que trata. Neste trabalho, o *solver* a ser estudado é o Algencan, que trata de resolver problemas de programação não linear. Contudo, esse *solver* está codificado na linguagem de programação Fortran, fazendo com que o usuário seja obrigado a codificar o problema nessa linguagem. Nesse sentido, é de interesse criar interoperabilidade com outras linguagens de programação compatíveis com Fortran. Com isso, usuários podem escolher qual linguagem utilizar e, inclusive, usufruir de recursos que simplifiquem a programação de um problema. Este trabalho tem por objetivo desenvolver interoperabilidade para o *software* Algencan com outras linguagens. Em um primeiro ponto, é demonstrado como realizar a configuração do Algencan para a sua correta utilização. Após essa etapa, são descritas as principais funções obrigatórias que este método computacional utiliza, assim como a resolução de um problema de otimização não linear. Em seguida, são demonstradas as interoperabilidades implementadas com as linguagens C, AMPL, Python e R.

Palavras-chaves: algencan; interoperabilidade; otimização.

Abstract

Optimization problems are problems that don't have polynomial time algorithms for solving them in the general case. For this reason, solvers are used, which are software that use algorithms that converge to an approximate solution of the optimization problem in question. In this work, the solver to be studied is Algencan, which deals with solving nonlinear programming problems. However, this solver is coded in the Fortran programming language, making the user to be forced to code the problem in that language. In this sense, it is of interest to create interoperability in other programming languages that have compatibility with Fortran. With this, users can choose which language to use and even take advantage of resources that simplify the programming of a problem. This work has the objective of develop interoperability for the Algencan software with other languages. In a first point, it is demonstrated how to configure Algencan for its correct use. After this step, the main mandatory functions that this computational method uses are described, as well as the resolution of a nonlinear optimization problem. Then, the interoperability implemented for the C, AMPL, Python and R languages are demonstrated.

Key-words: algencan; interoperability; optimization.

Lista de ilustrações

Lista de códigos

Lista de abreviaturas e siglas

AMPL	A Mathematical Programming Language - Uma Linguagem de Programação Matemática
BLAS	Basic Linear Algebra Subprogram - Subprograma Básico de Álgebra Linear
COO	Coordinate List - Lista Coordenada
CSR	Compressed Sparse Row - Linha Esparsa Comprimida
HSL	Harwell Subroutine library - Biblioteca de subrotinas Harwell
KKT	Karush-Kuhn-Tucke
LTS	Long Term Support - Suporte de Longo Prazo
PNL	Nonlinear Programming - Programação Não Linear
TANGO	Trustable Algorithms for Nonlinear General Optimization - Algoritmos Confiáveis para Otimização Geral Não Linear
TCC1	Trabalho de Conclusão de Curso 1
TCC2	Trabalho de Conclusão de Curso 2
USP	Universidade de São Paulo

Lista de símbolos

$\mathcal{L}$	Letra que representa a função de Lagrange
λ	Lambda
ρ	Letra grega minúscula rho
μ	Letra grega minúscula mu
γ	Letra grega minúscula gama
Ω	Letra grega maiúscula Omega
∇	Símbolo nabla que representa o operador gradiente
$\in$	Pertence
$\cap$	Intersecção
$\mathbb{R}$	Representação dos números reais
∂	Símbolo matemático que representa a derivada parcial

Sumário

1 Introdução

Os problemas de otimização passaram a receber grande importância, sendo utilizados em diversos projetos, tais como: determinação do nível mais econômico de uma fábrica, ponto da órbita de um cometa mais próximo da Terra, entre outros (CAVASOTTI, 2018). Problemas de otimização consistem em minimizar ou maximizar funções específicas. Segundo Cavasotti (2018), a otimização não linear envolve uma função que deve ser maximizada (ou minimizada) sujeita a um conjunto de restrições, em que pelo menos uma destas deve ter a característica de ser não linear.

Como não existe um algoritmo de tempo polinomial conhecido capaz de encontrar soluções para um problema de otimização no caso geral, são elaborados algoritmos iterativos que convergem para uma solução aproximada do problema. Como tais algoritmos são complexos e impraticáveis de serem executados manualmente, é comum o desenvolvimento de *software* robustos e eficientes que implementam esses algoritmos. Tais *software* são chamados de *solvers*. Neste trabalho, será levado em conta o *solver* Algencan.

O *software* Algencan é um método computacional para resolução de problemas de PNL (Programação Não Linear) (MARTíNEZ, 2006). Para utilizar o Algencan, é necessário que o usuário programe o problema de interesse. Essa programação consiste em escrever rotinas de avaliação que calculem a função objetivo e as restrições, bem como suas derivadas de primeiro e segundo grau.

Contudo, caso um usuário opte pelo uso do *solver* Algencan, o mesmo deve codificar os seus problemas na mesma linguagem em que o Algencan está escrito, isto é, em Fortran. Dessa forma, isso torna o uso desse *solver* bastante restrito. Com isso, surge o interesse no desenvolvimento de interoperabilidade com outras linguagens de programação ou modelagem matemática. No contexto deste trabalho, será denominado "interface" a implementação da interoperabilidade com outra linguagem, isto é, um programa codificado em uma determinada linguagem de programação que consegue chamar outros programas que estão escritos em diferentes linguagens.

1.1 Objetivos

O objetivo geral deste trabalho é construir interfaces que possibilitem a utilização do *software* Algencan em diferentes linguagens de programação e de modelagem. Esse objetivo é detalhado nos seguintes objetivos específicos:

- **OE01** - Desenvolver um manual de como instalar e configurar o *software* Algencan;

- **OE02** - Construir uma interface na linguagem de programação C;
- **OE03** - Construir uma interface na linguagem de modelagem algébrica AMPL;
- **OE04** - Construir uma interface na linguagem de programação Python;
- **OE05** - Construir uma interface na linguagem e ambiente de computação gráfica e estatística R.

1.2 Organização desta monografia

Este trabalho está estruturado em 5 capítulos. O Capítulo 1 mostra uma visão geral dos conceitos que contextualizam este trabalho. Além disso, é apresentado o seu objetivo geral e os seus objetivos específicos.

O Capítulo 2 apresenta os principais conceitos que envolvem este trabalho e como eles se relacionam. Nesse capítulo, está descrito uma base sobre programação não linear, métodos matemáticos para resolução de problemas de otimização, matriz esparsa e interoperabilidade com a linguagem C.

O Capítulo 3 apresenta o *software* Algencan, onde baixá-lo, como configurá-lo, suas funções obrigatórias e, por fim, é descrito a resolução de um problema de otimização não linear.

O Capítulo 4 apresenta as interfaces desenvolvidas para o presente trabalho descrevendo, assim, como funcionam, como configurá-las e como executá-las.

O Capítulo 5 apresenta as considerações finais do presente trabalho.

2 Referencial Teórico

Neste capítulo são apresentados os conceitos base para o entendimento e realização deste trabalho: Programação Não Linear (Seção 2.1), Métodos Matemáticos (Seção 2.2), Matriz Esparsa (Seção 2.3) e Interoperabilidade com C (Seção 2.4).

2.1 Programação não linear

Programação Não Linear (PNL) é uma técnica de resolução de problemas de otimização na qual muitas restrições apresentam a característica de não-linearidade. Em um problema de otimização genérico, é esperado o cálculo de máximos ou mínimos de uma função objetivo, que possui um conjunto de variáveis e um conjunto de restrições de igualdade e desigualdade que devem ser satisfeitas (SILVA et al., 2008).

Um problema de programação não linear possui a seguinte estrutura:

$$\begin{aligned} Min(ou \quad max) \quad & f(x) \\ sujeito \quad a \quad & h_j(x) = 0 \quad para \quad cada \quad j \in \{1, \ldots, m\}, \\ & g_i(x) \leq 0 \quad para \quad cada \quad i \in \{1, \ldots, p\}, \\ & para \quad x \geq 0, \end{aligned} \tag{2.1}$$

onde $n, m, p \geq 0$, $x \in \mathbb{R}^n$ é um vetor n-dimensional e f, g_i, para $i \in \{1, \ldots, p\}$ e h_j, $j \in \{1, \ldots, m\}$ são funções escalares. As principais características de um PNL é que todas essas funções devem ser contínuas e diferenciáveis e ao menos uma delas deve ser não linear.

2.2 Métodos Matemáticos

Na programação matemática para otimização de um problema não linear, tem-se o objetivo de minimizar ou maximizar uma função $f(x)$, denominada função objetivo, sujeito a algumas restrições $g(x)$ e $h(x)$ (desigualdade e igualdade, respectivamente). Quando um determinado ponto x satisfaz a todas as restrições impostas inicialmente, diz-se que ele é viável. Denominamos x^* como ponto ótimo, quando este minimiza ou maximiza $f(x)$ (SANTOS et al., 2010).

Dessa forma, para problemas que apresentam uma não-linearidade e que possuem restrições, existem alguns métodos para solucioná-los, entre eles o Método do Lagrangiano e o Método do Lagrangiano Aumentado, que serão detalhados a seguir.

2.2.1 Método Lagrangiano

Para solucionar o problema de otimização (2.1), é criada uma função que correlaciona a função objetivo com as restrições, denominada função Lagrangiana (SANTOS et al., 2010). Essa função possui um termo $\lambda \in \mathbb{R}^{m+p}$ denominado multiplicador de Lagrange, o qual pode ser adicionado ou subtraído. Essa função é dada por:

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{j=1}^{p} \lambda_{m+j} g_j(x). \tag{2.2}$$

Na função Lagrangiana acima $\mathcal{L}$, se $f(x)$ é um ponto de máximo para o problema original, então existe um λ tal que (x, λ) é um ponto estacionário, ou seja, existe um ponto para o qual as derivadas parciais de $\mathcal{L}$ são iguais a zero (BAPTISTA, 2001). Essa metodologia é garantida pelas condições de Karush-Kuhn-Tucker (ou condições KKT), que são condições de primeira ordem para que uma solução de um problema de programação não linear seja ótima (LUENBERGER; YE, 1984).

Contudo, o método Lagrangiano apresenta alguns problemas, fazendo com que o processo de otimização seja comprometido. Esses problemas estão relacionados com o aumento do número de variáveis e pela exigência de garantia da existência de um multiplicador (BAPTISTA, 2001). Dessa forma, o Método Lagrangiano Aumentado é criado com o intuito de corrigir tais problemas.

2.2.2 Método Lagrangiano Aumentado

O Método do Lagrangiano Aumentado tem por objetivo em, um dado ponto inicial $x_k \in \mathbb{R}^n$, encontrar uma solução para um determinado problema, resolvendo uma sequência de problemas de otimização irrestritos ou com restrições fáceis. Além disso, ele corrige o problema de instabilidade do método que faz o parâmetro de penalidade crescer sem limites. Para isso, é introduzido uma penalidade quando as restrições não são satisfeitas. Por esse motivo, este método costuma ser mais estável numericamente (BUENO; SENNE; OLIVEIRA, 2019).

Em outras palavras, esse método foi desenvolvido para resolver problemas de programação não linear com restrições gerais, com a possibilidade de utilizar de métodos eficientes para a resolução de subproblemas com restrições parciais, isto é, subproblemas que apresentam um subconjunto do conjunto de restrições do problema principal (LOBATO, 2009). Para exemplificar, considere o problema principal abaixo:

$$\begin{aligned} &minimizar \quad f(x) \\ &sujeita \quad a \quad x \in \Omega \end{aligned} \tag{2.3}$$

O conjunto de restrições Ω do problema principal pode ser reescrito como $\Omega = \Omega_1 \cap \Omega_2$, no qual o conjunto Ω_2 é composto por restrições consideradas simples, ou seja, problemas da forma

$$\begin{aligned} &minimizar \quad f(x) \\ &sujeita \quad a \quad x \in \Omega_2 \end{aligned} \tag{2.4}$$

são mais simples de se resolver do que problemas na forma (2.3). Dessa forma, o método do Lagrangiano Aumentado tem por objetivo resolver iterativamente o problema (2.3) tirando proveito dos métodos capazes de resolver os problemas da forma (2.4) (ANDREANI et al., 2003).

Portanto, o método do Lagrangiano Aumentado consiste em dois principais passos:

1. Minimizar o Lagrangiano Aumentado no conjunto simples apropriado (definido como Ω_2);
2. Atualizar os multiplicadores de Lagrange e o parâmetro de penalidade.

Dessa forma, a função que representa o Método do Lagrangiano Aumentado é dada pela equação (2.5), na qual $f(x)$ representa a função objetivo, ρ representa o parâmetro de penalidade, λ e μ representam os multiplicadores de Lagrange e $h(x)$ e $g(x)$ representam as restrições de igualdade e desigualdade, respectivamente.

$$\mathcal{L}_\rho(x, \lambda, \mu) = f(x) + \frac{\rho}{2}\left[\left\| h(x) + \frac{\lambda}{\rho} \right\|_2^2 + \left\| \left(g(x) + \frac{\mu}{\rho} \right)_+ \right\|_2^2\right] \tag{2.5}$$

2.3 Matriz Esparsa

Durante a resolução de um problema de otimização não linear, é comum a utilização de matrizes esparsas, já que os problemas envolvidos podem possuir muitas variáveis. A Figura 1 mostra um exemplo de matriz esparsa. Stoer e Bulirsch (2002) afirma que uma matriz esparsa é uma matriz cuja maioria dos seus elementos são iguais a zero, isto é, são nulos. Segundo Saad (2003), uma matriz esparsa é aquela que se pode obter vantagens pelo fato da mesma possuir uma grande quantidade de elementos nulos, visto que tais elementos não precisam ser armazenados.

Dessa forma, existem vários formatos/técnicas que podem ser utilizadas para representar matrizes esparsas. Para este trabalho, será utilizado dois desses formatos, o CSR (*Compressed Sparse Row*), que está descrito na Subseção 2.3.1, e o COO (*Coordinate List*), que está descrito na Subseção 2.3.2.

Figura 1 – Matriz Esparsa

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -2 \\ 3 & 0 & 2 & 0 & 1 \\ 0 & -4 & 0 & 7 & 0 \\ 0 & -5 & 0 & 0 & 0 \\ 0 & -6 & 0 & 0 & 6 \end{bmatrix}$$

Fonte: adaptado de (STOER; BULIRSCH, 2002)

2.3.1 CSR

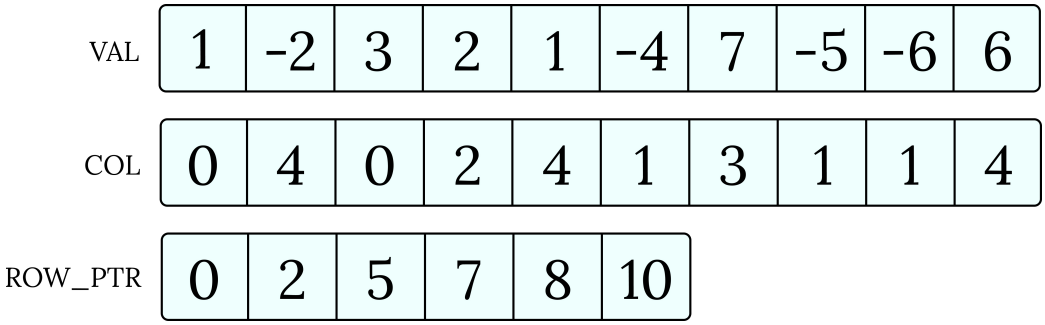
Compressed Sparse Row (CSR), *Compressed Row Storage* ou *Yale Format* (YF) é uma técnica para representação de matriz esparsa. Dessa forma, se esses elementos nulos forem armazenados na matriz, terá uma maior complexidade de espaço e um maior tempo gasto nas operações que forem necessárias.

O CSR é talvez um dos primeiros formatos de representação de matriz esparsa. Com ele, é possível obter uma maior eficiência de armazenamento e tempo. Esse formato propõe o uso de 3 vetores unidimensionais (VAL, COL, ROW_PTR) para representar uma matriz e a quantidade de elementos diferentes de zero é denotada por NNZ (ALAHMADI et al., 2020). Com isso, temos as seguintes propriedades:

- Uma dada matriz possui tamanho $m \times n$, no qual m representa a quantidade de linhas e n a quantidade de colunas;
- Os vetores VAL e COL possuem tamanho NNZ e contêm os valores diferentes de zero e os índices das colunas desses valores, respectivamente;
- O vetor ROW_PTR possui tamanho $m+1$ e divide o vetor VAL em linhas, indicando o índice de VAL e COL onde cada linha começa;
- O último elemento de ROW_PTR é NNZ.

Para uma melhor compreensão, será descrito um exemplo de como utilizar o formato de representação de matriz esparsa CSR. Considere a matriz esparsa de tamanho 5×5 demonstrada na Figura 1. Para representarmos essa matriz esparsa, colocamos todos os elementos não nulos no vetor VAL, linha por linha. Depois dessa etapa, informamos o índice da coluna da matriz de cada um desses elementos e os colocamos no vetor COL. Por fim, é informado no vetor ROW_PTR os índices do vetor VAL de onde começa cada uma das linhas. A representação de cada um desses 3 vetores pode ser vista na Figura 2.

Figura 2 – Formato CSR



Fonte: Autor

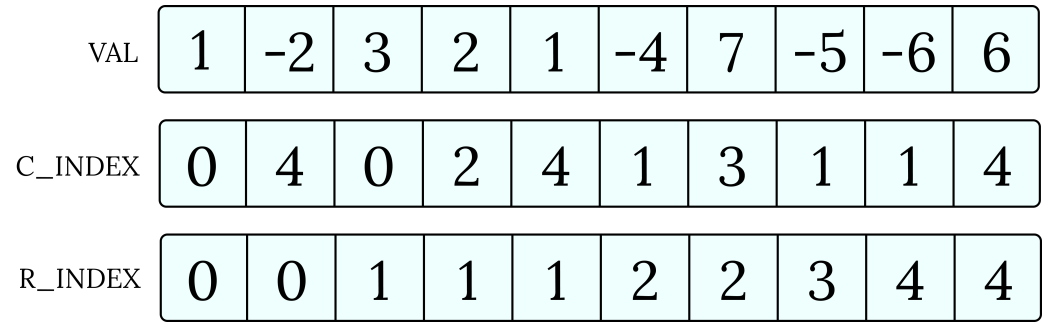
2.3.2 COO

Coordinate List (COO) é uma técnica para representação de matriz esparsa, assim como o formato CSR descrito na subseção 2.3.1. Uma matriz armazenada no formato COO consiste no uso de 3 vetores unidimensionais (VAL, C_INDEX, R_INDEX) de tamanho γ cada (DANG; SCHMIDT, 2012). Com isso, temos as seguintes propriedades:

- Uma dada matriz possui tamanho $m \times n$, no qual m representa a quantidade de linhas e n a quantidade de colunas;
- Os vetores VAL, C_INDEX e R_INDEX são responsáveis por armazenar o valor, o índice da coluna e o índice da linha dos elementos não nulos, respectivamente;
- γ representa a quantidade de elementos não nulos presente na matriz.

Para uma melhor compreensão, será descrito um exemplo de como utilizar o formato de representação de matriz esparsa COO. Ainda levando em consideração a matriz esparsa de tamanho 5×5 demonstrada na Figura 1, colocamos todos os elementos não nulos da matriz no vetor VAL. Após essa etapa, informamos o índice da coluna e o índice da linha de cada um desses elementos e os colocamos nos vetores C_INDEX e R_INDEX, respectivamente. A representação de cada um desses 3 vetores pode ser vista na Figura 3.

Figura 3 – Formato COO



Fonte: Autor

2.4 Interoperabilidade com C

Segundo Orengo (2014), a interoperabilidade entre linguagens significa que é possível acessar bibliotecas, métodos e outros, de uma linguagem de programação para a outra. Dessa forma, considerando que o Algencan é escrito na linguagem Fortran, o mesmo, desde a sua versão de 2003, possui uma maior interoperabilidade com a linguagem C, ou seja, bibliotecas escritas em Fortran podem ser acessadas em C. Para este trabalho, foi utilizado o compilador `gfortran`.

Desde o Fortran 2003, existe uma maneira padronizada de gerar procedimentos e declarações que são interoperáveis com a linguagem C. O atributo `bind(C)` é adicionado para informar ao compilador que um método, por exemplo, deve ser interoperável com C (GNU, 2004a). Além disso, é possível utilizar um módulo intrínseco chamado `iso_c_binding` que fornece um conjunto de constantes e procedimentos nomeados que auxiliam na programação com linguagens mistas (GNU, 2004b).

Nos Códigos 1 e 2, pode ser visualizado o uso do `bind(C)` e do `iso_c_binding` como demonstração da interoperabilidade entre as linguagens, na qual o código em C chama a função `add()` que está codificada em Fortran.

Código 1 – Chamar Fortran de C (`add_c.c`)

```c
#include <stdio.h>

typedef struct {
  int a, b, c;
} pdata_type;

int main() {
  extern void add(pdata_type *pdata);
  pdata_type pdata;
  pdata.a = 3;
  pdata.b = 5;
  add(&pdata);
  printf("c = %d\n", pdata.c);
  return 0;
}
```

Código 2 – Chamar Fortran de C (`add_f.f90`)

```
module shared_functions
  use iso_c_binding, only : c_int
  implicit none

```

```
5   type, bind(C) :: c_struct
6     integer(c_int) :: a, b, c
7   end type
8
9   contains
10
11  subroutine add(pdata) bind(C)
12    type(c_struct) :: pdata
13    pdata%c = pdata%a + pdata%b
14  end subroutine add
15 end module shared_functions
```

Para realizar a compilação do Código 1, execute o seguinte comando apresentado na Figura 4.

Figura 4 – Compilação do arquivo `add_c.c`

```
fellipe in TCC/interoperability/c_call_fortran
→ gcc -c add_c.c
```

Fonte: Autor

Para realizar a compilação do Código 2, execute o seguinte comando apresentado na Figura 5.

Figura 5 – Compilação do arquivo `add_f.f90`

```
fellipe in TCC/interoperability/c_call_fortran
→ gcc -o add add_f.o add_c.o -lgfortran
```

Fonte: Autor

Para realizar a geração do binário `add`, execute o seguinte comando apresentado na Figura 6.

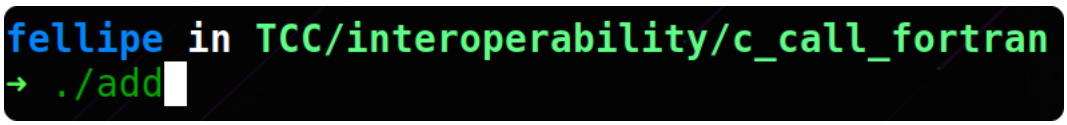
Figura 6 – Geração do binário `add`

```
fellipe in TCC/interoperability/c_call_fortran
→ gcc -o add add_f.o add_c.o -lgfortran
```

Fonte: Autor

Para realizar a execução do binário `add`, execute o seguinte comando apresentado na Figura 7.

Figura 7 – Execução do binário `add`



Fonte: Autor

Nos Códigos 3 e 4, pode ser visualizado o inverso, no qual o código em Fortran consegue realizar a operação de subtração que foi definida no código em C.

Código 3 – Chamar C de Fortran (`sub_c.c`)

```c
#include "stdio.h"

void sub(int *a, int *b, int *c) {
  *c = *a - *b;
}

int main() {
  extern void subtraction(void(*sub)(int *a, int *b, int *c))
   ;
  subtraction(sub);
  return 0;
}
```

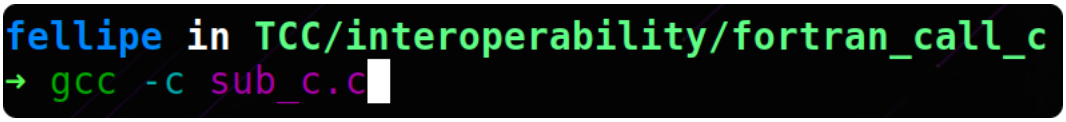
Código 4 – Chamar C de Fortran (`sub_f.f90`)

```fortran
module shared_functions
  implicit none

  interface
    subroutine fsub(a, b, c) bind(C)
      integer, intent(in) :: a, b
      integer, intent(out) :: c
    end subroutine fsub
  end interface

  contains

  subroutine subtraction(sub_c) bind(C, name='subtraction')
    procedure(fsub) :: sub_c
    integer :: a, b, c
    a = 5
    b = 3
```

```
18      call sub_c(a, b, c)
19      print *, 'c = ', c
20    end subroutine subtraction
21  end module shared_functions
```

Para realizar a compilação do Código 3, execute o seguinte comando apresentado na Figura 8.

Figura 8 – Compilação do arquivo `sub_c.c`



Fonte: Autor

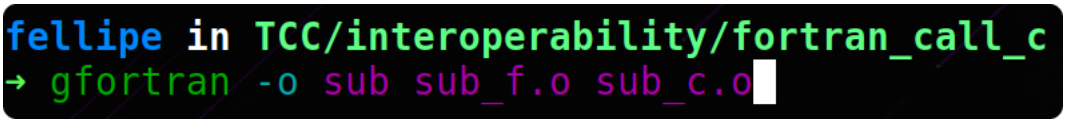
Para realizar a compilação do Código 4, execute o seguinte comando apresentado na Figura 9.

Figura 9 – Compilação do arquivo `sub_f.f90`

Fonte: Autor

Para realizar a geração do binário `sub`, execute o seguinte comando apresentado na Figura 10.

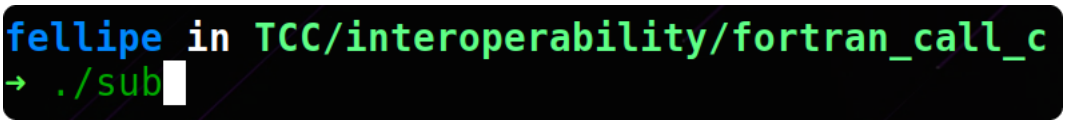
Figura 10 – Geração do binário `sub`



Fonte: Autor

Para realizar a execução do binário `sub`, execute o seguinte comando apresentado na Figura 11.

Figura 11 – Execução do binário `sub`



Fonte: Autor

3 Algencan

O Algencan é um *software* que foi desenvolvido pelos professores Ernesto Birgin da Universidade de São Paulo e José Mário Martínez da Universidade Estadual de Campinas. Esse *software* está escrito na linguagem de programação Fortran e tem o intuito de resolver problemas de otimização não linear de grande porte com tempo de computador moderado. O Algencan faz parte do projeto TANGO (Trustable Algorithms for Nonlinear General Optimization) e pode ser acessado livremente no site da USP[1]. Atualmente, a versão mais atual disponível desse *software* é a versão 3.1.1. Porém, para o estudo do presente trabalho de conclusão de curso será utilizado a versão 4.0.0, que encontra-se ainda em desenvolvimento e é fruto de um estudo de complexidade em programação não linear (BIRGIN; MARTíNEZ, 2020).

3.1 Instalação e Configuração

Para o uso do *software* Algencan, será necessária a instalação de alguns pacotes e bibliotecas, além da configuração de algumas variáveis de ambiente. Além disso, é possível utilizar a ferramenta `make` (subseção 3.1.5) para realizar as compilações necessárias de forma mais fácil e prática. Dessa forma, caso o `make` seja utilizado, poderá ser pulado as subseções 3.1.2 (Basic Linear Algebra Subprograms), 3.1.3 (Harwell Subroutine library) e 3.1.4 (Algencan), e prosseguir para a subseção 3.1.5 (Make).

A instalação e configuração da presente seção foi feita em um sistema operacional Ubuntu 22.04.1 (LTS). Portanto, a instalação e configuração em outros sistemas operacionais e/ou versões podem sofrer alterações.

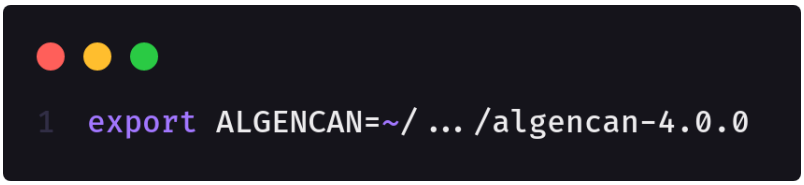
3.1.1 Variáveis de ambiente

Variáveis de Ambiente são valores dinâmicos que o sistema operacional ou outro *software* podem utilizar para determinar informações específicas para o computador. Elas são sobrescritas por argumentos de linha de comando ao utilizarmos um terminal como, por exemplo, o `shell`. Existem alguns tipos de variáveis de ambiente, como as de sistema e as de usuário. A maioria das variáveis de ambiente de sistema apontam para locais importantes como, por exemplo, binários de bibliotecas. Com isso, *software* que necessitam dessas bibliotecas conseguem acessar seus binários através das variáveis para funcionarem corretamente.

[1] <https://www.ime.usp.br/~egbirgin/tango/codes.php>

Por conveniência, para a utilização do *software* Algencan, será definida uma variável de ambiente como mostra a Figura 12:

Figura 12 – Variável de Ambiente - Algencan



Fonte: Autor

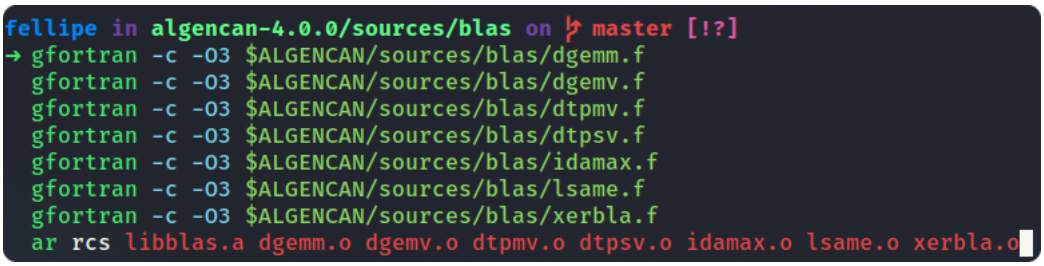
A Figura 12 mostra que é necessário a configuração da variável de ambiente ALGENCAN, a qual deve apontar para o caminho onde o *software* Algencan se encontra.

3.1.2 Basic linear algebra subprograms

BLAS (Basic Linear Algebra Subprograms) são rotinas de baixo nível que fornecem blocos de construção padrão para realizar operações de vetores e matrizes. O BLAS é altamente eficiente e amplamente disponível. Com isso, ele é extremamente útil no desenvolvimento de *software* de álgebra linear. Ele pode ser baixado no site do netlib[2] de forma gratuita. Após ele ser baixado, as seguintes etapas descritas a seguir auxiliam na correta configuração:

1. os arquivos baixados devem ser colocados no seguinte diretório:
   a) `$ALGENCAN/sources/blas/`.
2. uma sequência de comandos deve ser executada via linha de comando para compilar os arquivos BLAS, como mostra a Figura 13:

Figura 13 – Compilação do BLAS



Fonte: Autor

3. mova a biblioteca gerada `libblas.a` para o seguinte diretório:
   a) `$ALGENCAN/sources/blas/lib/`.

[2] <https://netlib.org/blas>

3.1.3 Harwell subroutine library

HSL (Harwell subroutine library) é uma biblioteca de software matemático que possui um conjunto de pacotes de última geração para computação científica que foi desenvolvida pelo *Computational Mathematics Group*. Essa biblioteca oferece um alto padrão de confiabilidade e é reconhecida internacionalmente como fonte de *software* numérico robusto e eficiente. Apesar dessa biblioteca ser paga, uma versão para pesquisa acadêmica pode ser baixada no site da *Science & Technology Facilities Council*[3]. Após ela ser baixada, as seguintes etapas descritas a seguir auxiliam na correta configuração:

1. os arquivos baixados devem ser colocados no seguinte diretório:

   a) `$ALGENCAN/sources/hsl/`.

2. uma sequência de comandos deve ser executada via linha de comando para compilar os arquivos HSL, como mostra a Figura 14:

Figura 14 – Compilação do HSL

```
fellipe in algencan-4.0.0/sources/hsl on  master [!?]
→ gfortran -c -O3 $ALGENCAN/sources/hsl/hsl_zd11d.f90
  gfortran -c -O3 $ALGENCAN/sources/hsl/hsl_ma57d.f90
  gfortran -c -O3 $ALGENCAN/sources/hsl/ma57ad.f
  gfortran -c -O3 $ALGENCAN/sources/hsl/mc34ad.f
  gfortran -c -O3 $ALGENCAN/sources/hsl/mc47ad.f
  gfortran -c -O3 $ALGENCAN/sources/hsl/mc59ad.f
  gfortran -c -O3 $ALGENCAN/sources/hsl/mc64ad.f
  gfortran -c -O3 $ALGENCAN/sources/hsl/mc21ad.f
  gfortran -c -O3 $ALGENCAN/sources/hsl/mc71ad.f
  gfortran -c -O3 $ALGENCAN/sources/hsl/fakemetis.f
  ar rcs libhsl.a hsl_zd11d.o hsl_ma57d.o ma57ad.o mc34ad.o mc47ad.o \
    mc59ad.o mc64ad.o mc21ad.o mc71ad.o fakemetis.o
```

Fonte: Autor

3. mova a biblioteca gerada `libhsl.a` para o seguinte diretório:

   a) `$ALGENCAN/sources/hsl/lib/`.

3.1.4 Algencan

Para compilar o algencan e criar a sua respectiva biblioteca, as seguintes etapas descritas a seguir auxiliam na correta configuração:

1. uma sequência de comandos deve ser executada via linha de comando para compilar os arquivos do algencan, como mostra a Figura 15:

[3] <https://www.hsl.rl.ac.uk/catalogue/ma57.html>

Figura 15 – Compilação do Algencan

```
fellipe in algencan-4.0.0/sources/algencan on  master [!?]
➜ gfortran -c -O3 -Wall -I$ALGENCAN/sources/hsl/inc $ALGENCAN/sources/algencan/lss.f90
  gfortran -c -O3 -Wall $ALGENCAN/sources/algencan/gencan.f90
  gfortran -c -O3 -Wall $ALGENCAN/sources/algencan/memev.f90
  gfortran -c -O3 -Wall $ALGENCAN/sources/algencan/feasgencan.f90
  gfortran -c -O3 -Wall $ALGENCAN/sources/algencan/newtkkt.f90
  gfortran -c -O3 -Wall $ALGENCAN/sources/algencan/algencan.f90
  ar rcs libalgencan.a lss.o gencan.o memev.o feasgencan.o newtkkt.o algencan.o
```

Fonte: Autor

2. mova a biblioteca gerada `libalgencan.a` para o seguinte diretório:

   a) `$ALGENCAN/sources/algencan/lib/`.

3. mova os módulos gerados `bmalgencan.mod`, `bmgencan.mod` e `bmfeasgencan.mod` para o seguinte diretório:

   a) `$ALGENCAN/sources/algencan/inc/`.

3.1.5 Make

O `make` é uma ferramenta de linha de comando que determina automaticamente quais arquivos precisam ser compilados, ou seja, quais foram alterados desde a última compilação ou que nunca foram compilados, e emite os comandos necessários para realizar tal ação. Para compilar e gerar as bibliotecas e módulos `libblas.a`, `libhsl.a`, `libalgencan.a`, `bmalgencan.mod`, `bmgencan.mod` e `bmfeasgencan.mod`, esteja no diretório `$ALGENCAN` e execute o seguinte comando especificado na Figura 16:

Figura 16 – Compilação utilizando o make

```
fellipe in algencan-4.0.0 on  master [!?]
➜ make
```

Fonte: Autor

3.2 Funções Obrigatórias

O *software* Algencan recebe uma série de parâmetros importantes e essenciais para que ele consiga ser executado com êxito. Dentre esses parâmetros, existem 5 funções que valem a pena serem mencionadas: `evalf`, `evalg`, `evalc`, `evalj` e `evalhl`. Essas funções não necessariamente precisam possuir esses nomes. Abaixo pode ser visto com mais detalhes a descrição do que cada uma dessas funções devem executar:

- `evalf`: essa função é responsável por computar a função objetivo $f(x)$;

- `evalg`: essa função é responsável por computar o gradiente da função objetivo, denotado por $\nabla f(x)$. O gradiente de uma função $f(x)$ nada mais é que a coleção de suas derivadas parciais dispostas em uma matriz de tamanho $n \times 1$, no qual n é a quantidade de variáveis da função em questão;

- `evalc`: essa função é responsável por computar todas as $m + p$ restrições, no qual m representa a quantidade de restrições de igualdade e p representa a quantidade de restrições de desigualdade. A coleção dessa soma das restrições é disposta em um vetor c, que pode ser definido como $c \in \mathbb{R}^{m+p}$, de tal forma que as m restrições de igualdade apareçam primeiro;

- `evalj`: essa função é responsável por computar o Jacobiano das restrições. O Jacobiano das restrições é uma matriz $(m+p) \times n$ que contém, em cada linha, o transposto do gradiente de cada uma das restrições h_i, $i \in \{1, \ldots, m\}$ e $g_j, j \in \{1, \ldots, p\}$, ou seja, é uma matriz da forma

$$J = \begin{pmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial x_1} & \frac{\partial h_m}{\partial x_2} & \cdots & \frac{\partial h_m}{\partial x_n} \\ \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_p}{\partial x_1} & \frac{\partial g_p}{\partial x_2} & \cdots & \frac{\partial g_p}{\partial x_n} \end{pmatrix}_{(m+p)\times n}.$$

  Para realizar esta ação, essa função utiliza de uma técnica chamada CSR (Compressed Sparse Row) para representar de uma forma melhor uma matriz esparsa. Essa técnica foi descrita na Subseção 2.3.1.

- `evalhl`: essa função é responsável por computar a Hessiana do Lagrangiano $\nabla^2 \mathcal{L}(x, \lambda) \in \mathbb{R}^{n \times n}$, que pode ser definida por

$$\nabla^2 \mathcal{L}(x, \lambda) = \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x) + \sum_{j=1}^{p} \lambda_{m+j} \nabla^2 h_j(x),$$

  onde $\nabla^2 f(x), \nabla^2 g_i(x)$ e $\nabla^2 h_j(x)$ são as matrizes Hessianas (ou seja, das derivadas parciais de segunda ordem) das funções $f(x)$, $g_i(x)$, $i = 1, 2, \ldots, m$, e $h_j(x)$, $j = 1, 2, \ldots, p$ e $\lambda \in \mathbb{R}^{m+p}$ é o chamado *vetor de multiplicadores de Lagrange*. Esse vetor λ é calculado iterativamente pelo Algencan e é passado como parâmetro para a função `evalhl`. Para realizar esta ação, essa função utiliza de uma técnica chamada COO (Coordinate List) para representar de uma forma melhor uma matriz esparsa. Essa técnica foi descrita na Subseção 2.3.2.

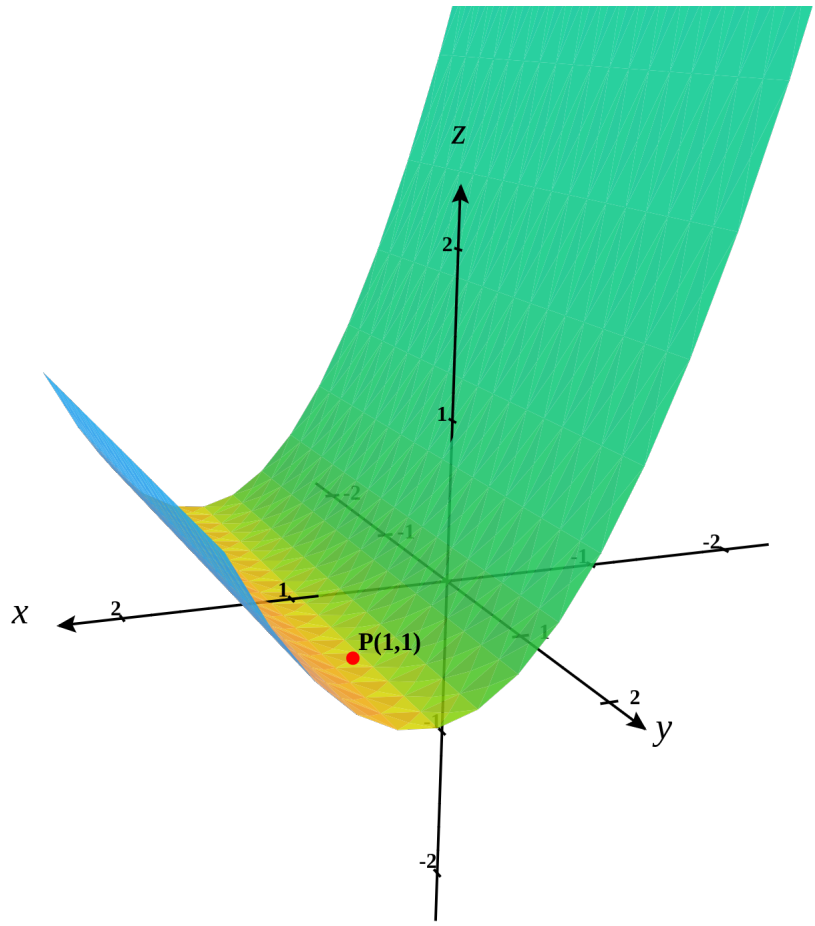
3.3 Resolução de um Problema

Para compreender a utilização do Algencan, será demonstrado passo a passo a resolução de um problema de otimização não linear. Dessa forma, considere o exemplo abaixo:

$$\begin{aligned} &Minimizar && f(x_1, x_2) = \Big(1 - x_1\Big)^2 \\ &sujeito \quad a && h_1(x_1, x_2) : 10\Big(x_2 - x_1^2\Big) \\ &com && x_0 = (-1.2, 1), \\ & && f(x_0) = 4.84 \end{aligned} \tag{3.1}$$

A função $f(x_1, x_2)$ pode ser visualizada graficamente pela Figura 17, na qual o eixo z representa $f(x_1, x_2)$, o eixo x representa x_1, o eixo y representa x_2 e o ponto em vermelho representado por $P(1, 1)$ é a solução esperada.

Figura 17 – Representação da função $f(x_1, x_2)$



Fonte: Autor

Dessa forma, temos 5 funções obrigatórias, as quais foram especificadas na Seção 3.2, que o *software* Algencan precisa que sejam definidas para que seja possível resolver

o problema de otimização. A seguir, é demonstrado o passo a passo para a definição de cada uma delas:

(a) `evalf`: $f(x_1, x_2) = (1 - x_1)^2$;

(b) `evalg`: $g(x_1, x_2) = \nabla f(x_1, x_2) = \begin{pmatrix} -2 + 2x_1 \\ 0 \end{pmatrix}$;

(c) `evalc`: para o presente problema, temos apenas 1 restrição de igualdade e 0 de desigualdade, ou seja, $m = 1$, sendo que m representa a quantidade de restrições de igualdade. Assim, $c(x_1, x_2) = 10\Big(x_2 - x_1^2\Big)$;

(d) `evalj`: a Jacobiana das restrições é dada pela seguinte matriz J de tamanho 1×2: $J = \begin{pmatrix} -20.0x_1 & 10 \end{pmatrix}_{1\times 2}$;

(e) `evalhl`: a Hessiana do Lagrangiano é dada por:

$$\nabla^2 \mathcal{L}(x, \lambda) = \begin{pmatrix} 2.0 - 20.0\lambda & 0 \\ 0 & 0 \end{pmatrix}.$$

Dessa forma, com as 5 rotinas de avaliação definidas, o Algencan irá resolver esse problema de otimização não linear encontrando, assim, a solução esperada representada pelo ponto $P(1, 1)$. Os códigos em Fortran para a utilização de cada uma das 5 rotinas citadas acima, podem ser visualizados no Anexo A.

4 Implementação

Neste capítulo são apresentados os resultados obtidos no presente trabalho, que é o desenvolvimento das interfaces. Nas seguintes seções, é descrito as interfaces, seus desenvolvimentos e exemplos de uso.

4.1 C

A linguagem C tem sido utilizada com sucesso em problemas de programação, desde sistemas operacionais, planilhas de texto, até sistemas expertos, e hoje em dia, estão disponíveis compiladores eficientes para máquinas de todo tipo de capacidade de processamento (COCIAN, 2004).

Dessa forma, como a linguagem C possui interoperabilidade com a linguagem Fortran (linguagem a qual o *software* Algencan está codificado), é possível criar uma interface com C para esse *solver*. Com isso, na interface com C também é necessário ter as mesmas 5 funções obrigatórias descritas na Seção 3.2, pois o Algencan as esperam como parâmetro.

4.1.1 Estrutura da Interface

Para a construção da interface é necessário apenas 1 arquivo:

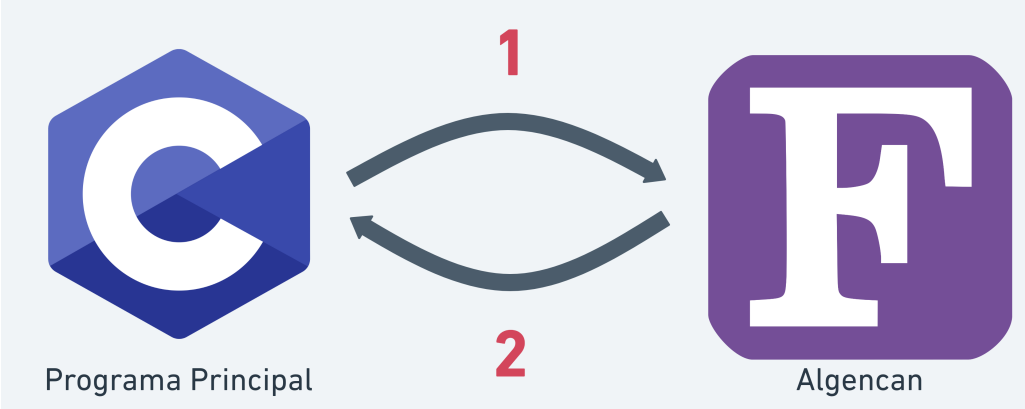
- `run_algencan.c`: programa principal escrito na linguagem C que faz a comunicação entre C-Fortran realizando, assim, a resolução do problema de otimização.

O fluxo de execução da interface para C pode ser visualizado na Figura 18, no qual as numerações representam as seguintes etapas:

- **Etapa 1**: o programa principal `algencanma.c` chama o Algencan passando como argumento todos os valores iniciais, parâmetros e rotinas descritas para o problema que se deseja resolver;

- **Etapa 2**: representa o fim da execução do Algencan e o retorno das variáveis atualizadas que foram passadas como argumento na etapa anterior.

Figura 18 – Fluxo de execução da interface para C. Em vermelho estão identificadas as etapas na ordem em que são executadas

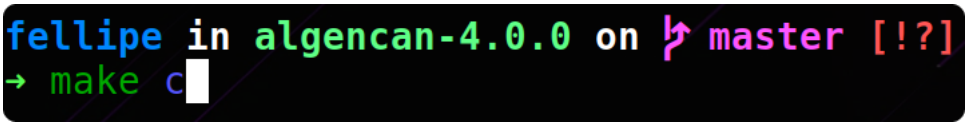


Fonte: Autor

4.1.2 Execução da Interface

Para a execução da interface com C, é necessário realizar a compilação utilizando o `make` como mostra a Figura 19.

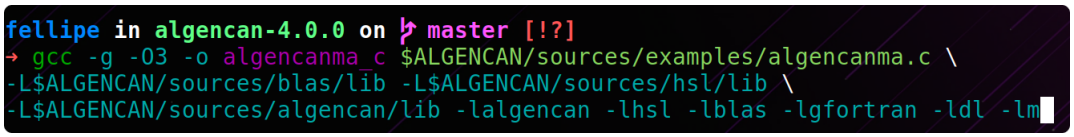
Figura 19 – Compilação da interface com C utilizando o `make`



Fonte: Autor

Alternativamente, a compilação sem a utilização do `make` pode ser feita conforme a Figura 20.

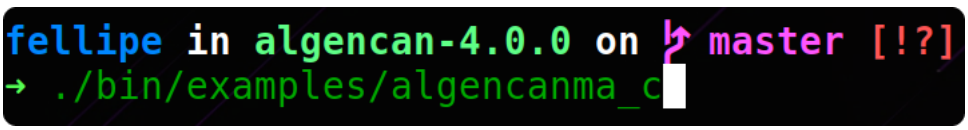
Figura 20 – Compilação da interface com C sem a utilização do `make`



Fonte: Autor

Feito a compilação, basta executar o binário `algencanma_c` como mostra a Figura 21.

Figura 21 – Execução da interface com C



Fonte: Autor

Dessa forma, para uma melhor descrição da interface para C, foi desenvolvido o código das rotinas de avaliação para solucionar o problema de otimização (3.1) descrito na Seção 3.3. As rotinas de avaliação encontram-se no Anexo B.

4.2 AMPL

O AMPL é uma linguagem para otimização em larga escala e para problemas de programação matemática em produção, distribuição, combinação, programação e muitas outras aplicações. Ele é combinado com uma notação algébrica familiar e um ambiente de comando interativo robusto e, com isso, o AMPL facilita a criação de modelos e o uso de uma ampla variedade de solucionadores (FOURER DAVID M. GAY, 2002). O *software* do AMPL pode ser baixado no site oficial[1].

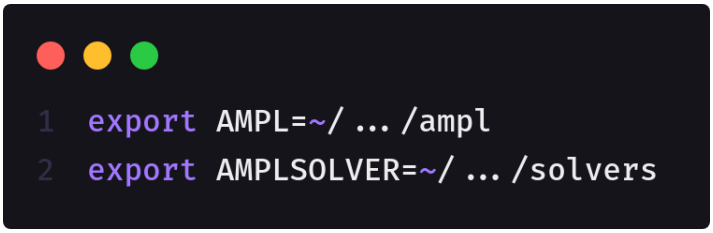
O AMPL por si só não consegue chamar um programa escrito na linguagem de programação Fortran. Por isso, a interface com AMPL precisa utilizar *wrappers* escritos na linguagem C para que a comunicação entre as linguagens seja possível. Um *wrapper* é uma entidade que encapsula outro item, ou seja, seu objetivo é converter dados em um formato compatível e/ou ocultar a entidade subjacente usando abstração (TECHOPEDIA, 2018).

Dessa forma, para computar e/ou calcular a função objetivo, o gradiente da função objetivo, as restrições, a Jacobiana das restrições e a Hessiana do Lagrangiano (como discutido nas funções obrigatórias do Algencan 3.2), é necessário utilizar algumas funções fornecidas no próprio site do AMPL[2], entre elas: `objval`, `objgrd`, `conval`, `congrd` e `sphes`. Abaixo é descrito o passo a passo de como baixar e configurar as bibliotecas:

- **Passo 1**: no site mencionado do AMPL, deve-se fazer o *download* do arquivo `solvers.tar`;
- **Passo 2**: deve-se descompactar o arquivo baixado;
- **Passo 3**: dentre os arquivos descompactados, haverá um arquivo *README* com as instruções de como configurar as bibliotecas baixadas;
- **Passo 4**: após finalizar a etapa anterior, um arquivo chamado `amplsolver.a` será criado. Agora, se faz necessário a criação de duas variáveis de ambiente como mostra a Figura 22:
    1. AMPL: deve apontar para o caminho onde o *software* AMPL foi baixado;
    2. AMPLSOLVER: deve apontar para o caminho onde o pacote de bibliotecas na linguagem C do AMPL foi baixado e configurado.

[1] <https://ampl.com>
[2] <https://ampl.com/REFS/HOOKING>

Figura 22 – Variáveis de Ambiente - AMPL



Fonte: Autor

4.2.1 Estrutura da Interface

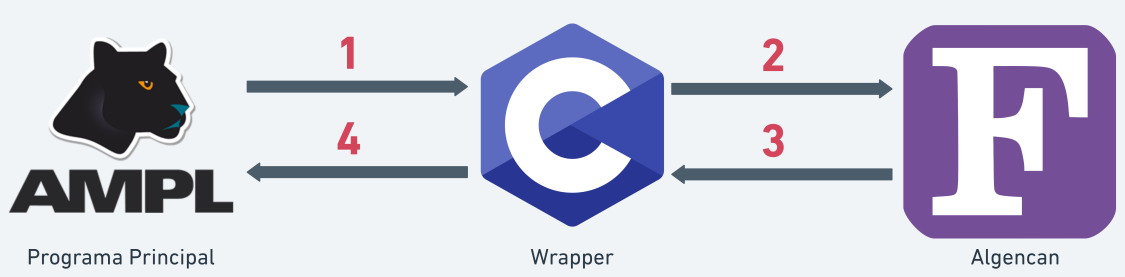
Para a construção da interface são necessários 3 arquivos:

- `ampl_wrapper.c`: *wrapper* escrito na linguagem C que faz a comunicação entre AMPL-C e C-Fortran;
- `problem.mod`: responsável pela descrição do problema a se resolver, no qual está incluso a definição da função $f(x)$ que se quer minimizar ou maximizar e as suas restrições;
- `algencan.run`: programa principal que chama o *wrapper* e inicia a resolução do problema de otimização.

Dessa forma, o fluxo de execução da interface para AMPL pode ser visualizado na Figura 23, no qual as numerações representam as seguintes etapas:

- **Etapa 1**: o programa principal `algencan.run` carrega a função e suas restrições, os valores iniciais dos $x's$ e chama o *wrapper* escrito na linguagem C;
- **Etapa 2**: o *wrapper* constrói as 5 rotinas de avaliação (`evalf`, `evalg`, `evalc`, `evalj` e `evalhl`) passando para elas as funções fornecidas pelo AMPL (`objval`, `objgrd`, `conval`, `congrd` e `sphes`). Depois, o *solver* Algencan é chamado;
- **Etapas 3 e 4**: essas etapas representam o fim da execução do Algencan e o retorno das variáveis atualizadas.

Figura 23 – Fluxo de execução da interface para AMPL. Em vermelho estão identificadas as etapas na ordem em que são executadas



Fonte: Autor

A rotina de avaliação `objval` é responsável por computar a função objetivo, como pode ser visto na Figura 24. A rotina `objval` recebe três argumentos: o argumento *nobj* é o objetivo, X é um vetor com os pontos iniciais e *nerror* controla o que acontece se a rotina detectar algum erro.

Figura 24 – Rotina de avaliação `objval()`

```
1  real objval(int nobj, real *X, fint *nerror)
```

Fonte: Autor

A rotina de avaliação `objgrd` é responsável por computar o gradiente da função objetivo, como pode ser visto na Figura 25. A rotina `objgrd` recebe quatro argumentos: o argumento *nobj* é o objetivo, X é um vetor com os pontos iniciais, G armazena o gradiente do objetivo *nobj* e *nerror* controla o que acontece se a rotina detectar algum erro.

Figura 25 – Rotina de avaliação `objgrd()`

```
1  void objgrd(int nobj, real *X, real *G, fint *nerror)
```

Fonte: Autor

A rotina de avaliação `conval` é responsável por computar as restrições do problema, como pode ser visto na Figura 26. A rotina `conval` recebe três argumentos: o argumento X representa o vetor com os pontos iniciais, R armazena as restrições avaliadas e *nerror* controla o que acontece se a rotina detectar algum erro.

Figura 26 – Rotina de avaliação `conval()`

```
1 void conval(real *X, real *R, fint *nerror)
```

Fonte: Autor

A rotina de avaliação `congrd` é responsável por calcular o gradiente das restrições, como pode ser visto na Figura 27. A rotina `congrd` recebe quatro argumentos: o argumento *ncon* representa o número da restrição, X é o vetor com os pontos iniciais, G armazena o gradiente das restrições e *nerror* controla o que acontece se a rotina detectar algum erro.

Figura 27 – Rotina de avaliação `congrd()`

```
1 void congrd(int ncon, real *X, real *G, fint *nerror)
```

Fonte: Autor

A rotina de avaliação `sphes` é responsável por avaliar e armazenar o triângulo superior esparso da matriz Hessiana da função Lagrangiana, como pode ser visto na Figura 28. A rotina `sphes` recebe quatro argumentos: H é a variável na qual é armazenado o triângulo superior, *nobj* é o objetivo, OW e Y são multiplicadores de Lagrange. Dessa forma, o armazenamento dos elementos em H é feito coluna por coluna. Com isso, a rotina `sphes` configura dois parâmetros globais: `sputinfo->hrownos[1:nnz]` que representa o número da linha a que corresponde o i-ésimo elemento de H e `sputinfo->hcolstarts[1:n]` que determina em que posição do vetor H começa a i-ésima coluna.

Figura 28 – Rotina de avaliação `sphes()`

```
1 void sphes(real *H, int nobj, real *OW, real *Y)
```

Fonte: Autor

4.2.2 Execução da Interface

Para a execução da interface com AMPL, é necessário realizar a compilação conforme a Figura 29.

Figura 29 – Compilação da interface com AMPL

```
fellipe in algencan-4.0.0 on  master [!?]
→ gcc -L$ALGENCAN/sources/blas/lib -L$ALGENCAN/sources/hsl/lib \
-L$ALGENCAN/sources/algencan/lib -o $AMPL/algencan -I$AMPLSOLVER \
$ALGENCAN/sources/interfaces/ampl/ampl_wrapper.c $AMPLSOLVER/amplsolver.a \
-lalgencan -lhsl -lblas -lgfortran -ldl -lm
```

Fonte: Autor

Feito a compilação, basta abrir a linha de comando do AMPL no diretório onde se encontra o programa principal `algencan.run` e executá-lo, conforme mostra a Figura 30.

Figura 30 – Execução da interface com AMPL

```
fellipe in algencan-4.0.0/sources/interfaces/ampl on  master [!?]
→ $AMPL/ampl
ampl: include algencan.run;
```

Fonte: Autor

Dessa forma, para uma melhor descrição da interface para AMPL, foi desenvolvido o código em AMPL e o seu *wrapper* em C para solucionar o problema de otimização (3.1) descrito na Seção 3.3. A descrição do problema em AMPL encontra-se no Anexo C e os *wrappers* de cada uma das rotinas de avaliação encontram-se no Anexo F.

4.3 PYTHON

Python é uma linguagem de programação *open-source* de alto nível, interpretada, orientada a objetos e que possui semântica dinâmica (COURSERA, 2022). O Python pode ser utilizado em várias situações como, por exemplo, desenvolvimento *web* (Django, Bottle), programação matemática e computacional (Orange, SymPy, NumPy), aplicativos para *desktop* (PyQt, PyWidgets), jogos (Pygame, Panda3D), entre outros (SOUSA, 2020).

Para a interface com Python, também será utilizado um *wrapper* escrito na linguagem C.

4.3.1 Estrutura da Interface

Para a construção da interface são necessários 3 arquivos:

- `problem.py`: responsável pela descrição do problema a se resolver, no qual está incluso a elaboração das 5 principais funções: `evalf`, `evalg`, `evalc`, `evalj` e `evalhl`, e funções que definem parâmetros e valores iniciais: `params` e `initial`;

- `py_wrapper.c`: *wrapper* escrito na linguagem C que faz a comunicação entre Python-C e C-Fortran;
- `run_algencan.py`: programa principal que carrega uma biblioteca compartilhada gerada pela compilação do arquivo `py_wrapper.c` e inicia a resolução do problema de otimização, conforme mostram os Códigos 5 e 6, respectivamente.

Código 5 – Carregamento da biblioteca compartilhada `solver.so` gerada pela compilação do arquivo `py_wrapper.c`

```
1  import ctypes as ct
2
3  funSolver = ct.PyDLL("./sources/interfaces/python/solver.so",
       mode=os.RTLD_LAZY)
4  funSolver.algencan_solver.argtypes = [
5          ct.c_char_p, ct.c_char_p, ct.c_char_p,
6          ct.c_char_p, ct.c_char_p, ct.c_char_p,
7          ct.POINTER(ct.c_double), ct.POINTER(ct.c_double),
8          ct.POINTER(ct.c_double), ct.POINTER(ct.c_double),
9          ct.POINTER(ct.c_double), ct.POINTER(ct.c_int),
10         ct.POINTER(ct.c_int), ct.POINTER(ct.c_int),
11         ct.POINTER(ct.c_int), ct.POINTER(ct.c_int),
12         ct.POINTER(ct.c_int), ct.POINTER(ct.c_int),
13         ct.POINTER(ct.c_double), ct.POINTER(ct.c_int),
14         ct.POINTER(ct.c_int), ct.POINTER(ct.c_int),
15         ct.POINTER(ct.c_int), ct.POINTER(ct.c_double),
16         ct.POINTER(ct.c_int)]
```

Código 6 – Execução da função `algencan_solver`

```
1  import ctypes as ct
2
3  funSolver.algencan_solver(
4          ct.c_char_p(str.encode(evalf_py.__name__)),
5          ct.c_char_p(str.encode(evalg_py.__name__)),
6          ct.c_char_p(str.encode(evalc_py.__name__)),
7          ct.c_char_p(str.encode(evalj_py.__name__)),
8          ct.c_char_p(str.encode(evalhl_py.__name__)),
9          ct.c_char_p(str.encode(params_py.__name__)),
10         f, csupn, ssupn, nlpsupn, bdsvio, outiter,
11         totiter, nwcalls, nwtotit, ierr, istop,
12         n, x, lind, uind, m, p, _lambda, jnnzmax)
```

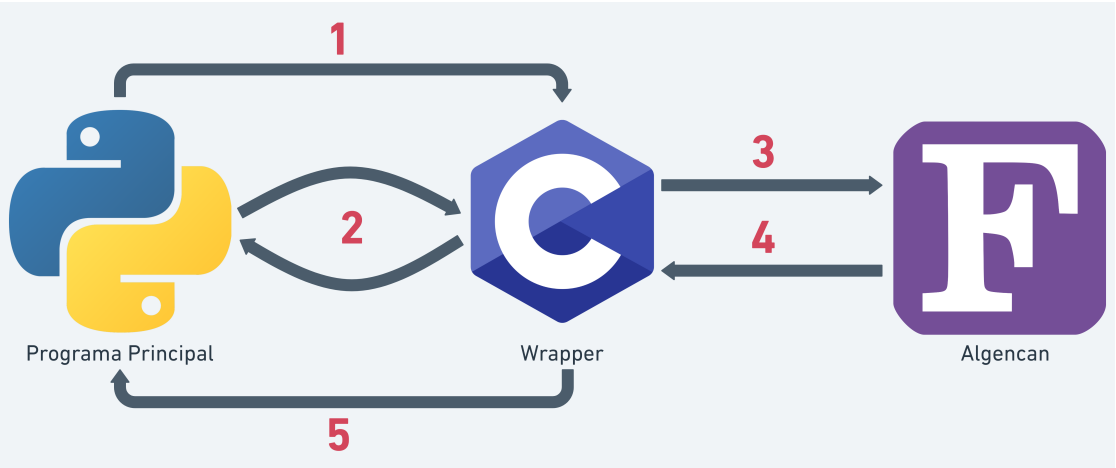
Na linha 3 do Código 5, é realizado o carregamento da biblioteca compartilhada. Na linha 4 do Código 5, é definido o tipo de dado de cada argumento na linguagem C que a função `algencan_solver` espera. E, por fim, na linha 3 do Código 6, é realizada a execução da função `algencan_solver`.

O nome das 5 principais rotinas de avaliação e das funções de definição de parâmetros e valores inicias são de livre escolha do usuário. Além disso, se faz necessário o uso de 2 bibliotecas para Python: `numpy` e `ctypes`. O `numpy` é necessário para a definição de alguns vetores e tipo de dados, enquanto que o `ctypes` permite que seja possível carregar uma biblioteca compartilhada e chamar um código escrito na linguagem C a partir do Python.

Dessa forma, o fluxo de execução da interface para Python pode ser visualizado na Figura 31, no qual as numerações representam as seguintes etapas:

- **Etapa 1**: o programa principal `run_algencan.py` chama o *wrapper* escrito na linguagem C passando como argumento uma série de variáveis, as quais serão obtidas com seus valores atualizados ao fim da execução do Algencan. Para realização dessa ação, é utilizado o `ctypes`;
- **Etapa 2**: o *wrapper* irá obter todos os valores iniciais, parâmetros e funções descritas para o problema a se resolver encontradas no arquivo `problem.py`;
- **Etapa 3**: o *wrapper* chama o Algencan, passando como argumento todos os valores e funções obtidas na etapa anterior;
- **Etapas 4 e 5**: essas etapas representam o fim da execução do Algencan e o retorno das variáveis atualizadas que foram passadas como argumento na Etapa 1.

Figura 31 – Fluxo de execução da interface para Python. Em vermelho estão identificadas as etapas na ordem em que são executadas



Fonte: Autor

4.3.2 API Python/C

Para a criação do *wrapper*, será utilizada a API Python/C[3]. Essa API fornece acesso ao interpretador Python em uma variedade de níveis, ou seja, seu objetivo é escrever módulos de extensão C que estendem o interpretador Python.

Para o desenvolvimento da interface com Python, foi utilizada da API Python/C as seguintes estruturas e métodos:

- `Py_Initialize`: inicializa o interpretador Python;
- `PyObject`: todos os tipos de objeto são extensões desse tipo. Este é um tipo que contém as informações que o Python precisa para tratar um ponteiro para um objeto como um objeto. Nada é realmente declarado como um `PyObject`, mas cada ponteiro para um objeto Python pode ser convertido em um `PyObject`;
- `PyUnicode_FromString`: constrói um objeto a partir de um *buffer* de caracteres. Esse *buffer* é representado pelo nome do arquivo que contém a resolução do problema em questão;
- `PyImport_Import`: realiza a importação do módulo do objeto construído pelo método `PyUnicode_FromString`, ou seja, realiza a importação do arquivo que contém a resolução de um problema;
- `PyModule_GetDict`: este método retorna o objeto de dicionário que implementa o *namespace* do módulo que foi importado por `PyImport_Import`;
- `PyDict_GetItemString`: tem a mesma função que o `PyModule_GetDict`, porém ao invés de receber como parâmetro um `PyObject`, recebe uma *string*. Essa *string* representa o nome de uma função que se quer importar a partir do arquivo que contém a resolução de um problema;
- `PyErr_Print`: imprime o erro obtido, caso o ocorra, após o fim da execução de algum método da API;
- `PyObject_CallObject`: realiza a chamada de uma função Python;
- `PyTuple_New`: retorna um novo objeto no formato de tupla;
- `PyTuple_SetItem`: insere uma referência do objeto na tupla;
- `PyTuple_GetItem`: retorna um objeto em uma determinada posição da tupla;
- `PyTuple_Size`: retorna o tamanho de uma tupla;

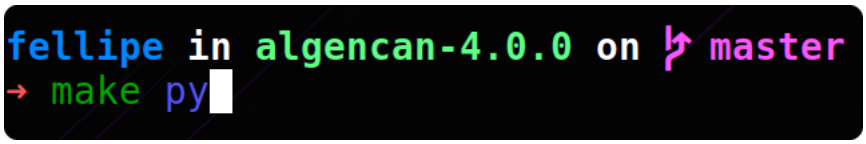
[3] <https://docs.python.org/pt-br/3/c-api/index.html>

- `PyFloat_FromDouble`: cria um objeto de ponto flutuante do Python a partir de um *double* em C;
- `PyFloat_AsDouble`: retorna um *double* em C a partir de um ponto flutuante em Python;

4.3.3 Execução da Interface

Para a execução da interface com Python, é necessário realizar a compilação utilizando o `make` como mostra a Figura 32. Essa compilação irá, dentre outras ações, criar uma biblioteca compartilhada chamada `solver.so`.

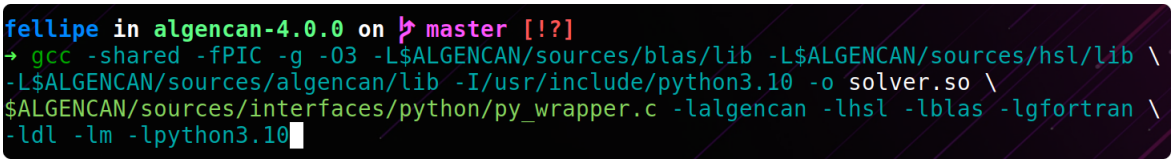
Figura 32 – Compilação da interface com Python utilizando o `make`



Fonte: Autor

Alternativamente, a compilação sem a utilização do `make` pode ser feita conforme a Figura 33.

Figura 33 – Compilação da interface com Python sem a utilização do `make`



Fonte: Autor

Feito a compilação, basta executar o arquivo `run_algencan.py` como mostra a Figura 34.

Figura 34 – Execução da interface com Python

Fonte: Autor

Para uma melhor descrição da interface para Python, foi desenvolvido o código das rotinas de avaliação para solucionar o problema de otimização (3.1) descrito na Seção 3.3. As rotinas de avaliação codificadas em Python encontram-se no Anexo D e os *wrappers* de cada uma das rotinas encontram-se no Anexo G.

4.4 R

R é uma linguagem e ambiente para computação gráfica e estatística. Essa linguagem fornece uma ampla variedade de técnicas de modelagem linear e não linear, testes estatísticos clássicos, análise de séries temporais, classificação, entre outros (R, 2022). Dessa forma, para tarefas de computação extensiva, há a possibilidade da escrita de códigos na linguagem C que podem ser vinculados e chamados em tempo de execução (TEAM, 2000).

Para a interface com R, também será utilizado um *wrapper* escrito na linguagem C.

4.4.1 Estrutura da Interface

Para a construção da interface são necessários 3 arquivos:

- `problem.R`: responsável pela descrição do problema a se resolver, no qual está incluso a elaboração das 5 principais funções: `evalf`, `evalg`, `evalc`, `evalj` e `evalhl`, e funções que definem parâmetros e valores iniciais: `params` e `initial`;
- `r_wrapper.c`: *wrapper* escrito na linguagem C que faz a comunicação entre R-C e C-Fortran;
- `run_algencan.R`: programa principal que carrega uma biblioteca compartilhada gerada pela compilação do arquivo `r_wrapper.c` e inicia a resolução do problema de otimização, conforme mostram os Códigos 7 e 8, respectivamente.

Código 7 – Carregamento da biblioteca compartilhada `r_wrapper.so` gerada pela compilação do arquivo `r_wrapper.c`

```
1 dyn.load("./sources/interfaces/r/r_wrapper.so")
```

Código 8 – Execução da função `algencan_solver`

```
1 result <- .C("algencan_solver",
2   as.character(substitute(evalf_r)), as.character(substitute(
    evalg_r)),
3   as.character(substitute(evalc_r)), as.character(substitute(
    evalj_r)),
4   as.character(substitute(evalhl_r)), as.character(substitute
    (params_r)),
5   f = f, csupn = csupn, ssupn = ssupn, nlpsupn = nlpsupn,
    bdsvio = bdsvio,
```

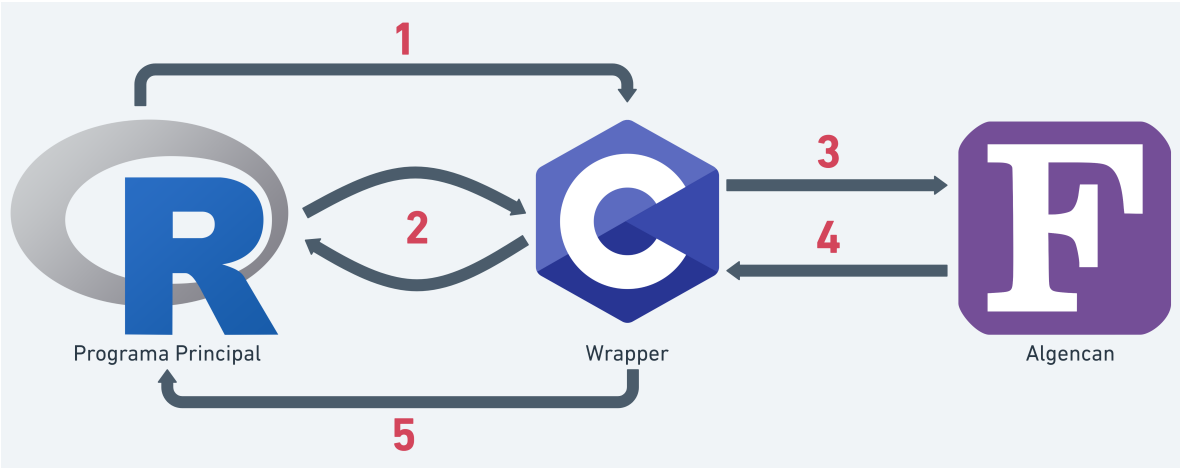
```
6   outiter = outiter, totiter = totiter, nwcalls = nwcalls,
     nwtotit = nwtotit,
7   ierr = ierr, istop = istop, n = n, x = x, lind = lind, uind
      = uind, m = m,
8   p = p, lambda = lambda, jnnzmax = jnnzmax
9 )
```

Na linha 1 do Código 7, é realizado o carregamento da biblioteca compartilhada. Na linha 1 do Código 8 é realizada a execução da função `algencan_solver`. Além disso, o nome das 5 principais rotinas de avaliação e das funções de definição de parâmetros e valores iniciais são de livre escolha do usuário.

Dessa forma, o fluxo de execução da interface para R pode ser visualizado na Figura 36, no qual as numerações representam as seguintes etapas:

- **Etapa 1**: o programa principal `run_algencan.R` chama o *wrapper* escrito na linguagem C passando como argumento uma série de variáveis, as quais serão obtidas com seus valores atualizados ao fim da execução do Algencan;
- **Etapa 2**: o *wrapper* irá obter todos os valores iniciais, parâmetros e funções descritas para o problema a se resolver encontradas no arquivo `problem.R`;
- **Etapa 3**: o *wrapper* chama o Algencan, passando como argumento todos os valores e funções obtidas na etapa anterior;
- **Etapas 4 e 5**: essas etapas representam o fim da execução do Algencan e o retorno das variáveis atualizadas que foram passadas como argumento na Etapa 1.

Figura 35 – Fluxo de execução da interface para R. Em vermelho estão identificadas as etapas na ordem em que são executadas



Fonte: Autor

4.4.2 API C interna do R

Para a criação do *wrapper* será utilizada a API C interna do R[4]. Essa API permite que seja criado tipos de dados na linguagem C e que sejam entendidos pela linguagem R, além de possibilitar a chamada de funções R a partir de C. Dessa forma, para o desenvolvimento da interface com R, foi utilizada da API C interna do R as seguintes estruturas e métodos:

- `SEXP`: é um tipo de dado comum em que todos os objetos R são armazenados no código em C. Todos os objetos R são expressões S e, portanto, cada função C criada deve retornar a `SEXP` como saída e receber uma `SEXP'S` como entrada;
- `Rf_protect`: usado para proteger objetos apontados por `SEXP`. Ele informa ao R que um determinado objeto está em uso e não deve ser excluído pelo GC (Garbage Collector). O GC libera memória automaticamente quando um objeto não é mais utilizado;
- `Rf_unprotect`: desprotege objetos apontados por `SEXP`;
- `Rf_lang`: são usados para configurar uma chamada de função, podendo ir de `Rf_lang1` a `Rf_lang6`, o qual o índice ao fim representa a quantidade de argumentos que esse método irá receber, sendo o primeiro argumento o nome da função definida em R, e os demais os seus argumentos;
- `Rf_install`: obtém um ponteiro para a função R que se quer executar;
- `Rf_mkstring`: converte uma dado do tipo `char*` para `SEXP`;
- `R_tryEval`: executa uma função R;
- `Rf_isReal`: esse método verifica se um determinado objeto é do tipo numérico;
- `Rf_isInteger`: esse método verifica se um determinado objeto é do tipo inteiro;
- `REAL`: função auxiliar utilizada para acessar a matriz C que armazena os dados em um vetor de `double`;
- `INTEGER`: função auxiliar utilizada para acessar a matriz C que armazena os dados em um vetor de `int`;
- `LENGTH`: informa a quantidade de elementos de uma lista em R que está representada por um `SEXP`;
- `VECTOR_ELT`: método utilizado para acessar um determinado elemento dentro de um vetor;

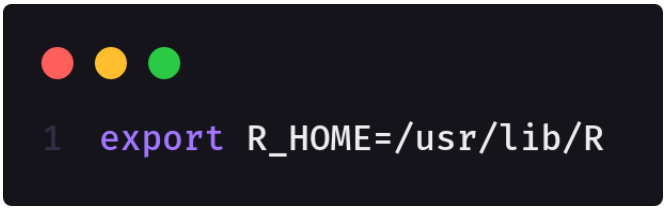
[4] <https://cran.r-project.org/doc/manuals/r-release/R-exts.html>

- `Rf_xlength`: informa o tamanho de um vetor;
- `Rf_allocVector`: esse método é responsável por criar um vetor R com um tamanho especificado;
- `REALSEXP`: representa o tipo de dado de um vetor em R e, nesse caso, é um vetor numérico;
- `Rf_ScalarInteger`: função auxiliar que transforma um escalar C em um vetor R de tamanho 1.

4.4.3 Execução da Interface

Para a correta execução da interface com R, se faz necessário ter o ambiente R baixado, além de configurar uma variável de ambiente como mostra a Figura 36. O Valor de `R_HOME` deve apontar para o local onde o *software* R foi baixado.

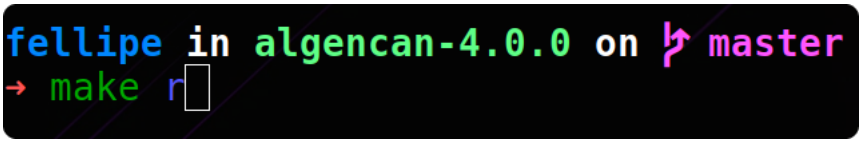
Figura 36 – Variável de ambiente para a interface com R



Fonte: Autor

Após a variável de ambiente estar configurada, é necessário realizar a compilação da interface com R utilizando o `make` como mostra a Figura 37. Essa compilação irá, dentre outras ações, criar uma biblioteca compartilhada chamada `r_wrapper.so`.

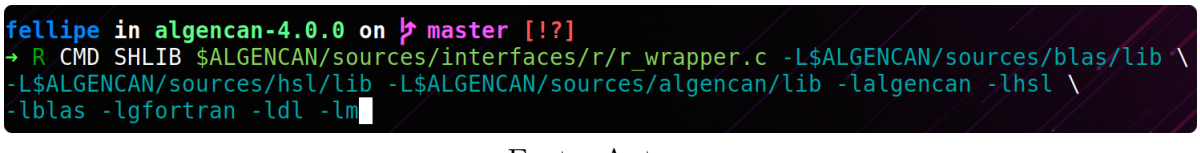
Figura 37 – Compilação da interface com R utilizando o `make`



Fonte: Autor

Alternativamente, a compilação sem a utilização do `make` pode ser feita conforme a Figura 38.

Figura 38 – Compilação da interface com R sem a utilização do `make`



Fonte: Autor

Feito a compilação, basta executar o arquivo `run_algencan.R` como mostra a Figura 39.

Figura 39 – Execução da interface com R

Fonte: Autor

Para uma melhor descrição da interface para R, foi desenvolvido o código das rotinas de avaliação para solucionar o problema de otimização (3.1) descrito na Seção 3.3. As rotinas de avaliação codificadas em R encontram-se no Anexo E e os *wrappers* de cada uma das rotinas encontram-se no Anexo H.

5 Considerações finais

Este projeto se propôs desenvolver quatro interfaces que consigam se comunicar com o *software* Algencan: uma para a linguagem de programação C, uma para a linguagem de modelagem algébrica AMPL, uma para a linguagem de programação Python e uma para linguagem e ambiente para computação estatística e gráficos R.

Os principais conceitos envolvidos neste trabalho sobre programação não linear, métodos matemáticos (Lagrangiano e Lagrangiano Aumentado), matriz esparsa e interoperabilidade com C, são apresentados no Capítulo 2. Neste trabalho, a versão do Algencan que foi utilizada foi a 4.0.0, a qual ainda será lançada futuramente (Capítulo 3). As interfaces para C, para AMPL, para Python e para R foram desenvolvidas, mais detalhes podem ser observados no Capítulo 4.

Como o *software* Algencan é um *solver* robusto, o seu entendimento de como funciona, suas principais funções obrigatórias (Seção 3.2) e de como configurá-lo (Seção 3.1) é de suma importância para que, assim, possa ser feito o desenvolvimento de interfaces para outras linaguagens.

Dessa forma, ao decorrer deste trabalho, percebeu-se que a utilização do Algencan em sua forma pura (linguagem de programação Fortran) torna o seu uso restrito, pois os usuários que optam por utilizá-lo, necessitam codificar os seus problemas de otimização na linguagem Fortran. Portanto, é interessante a construção de interfaces em diferentes linguagens de programação e/ou modelagem para que, assim, seja possível a utilização do *solver* Algencan em outras linguagens.

O principal objetivo do presente trabalho foi construir interfaces que possibilitassem a utilização do *software* Algencan em diferentes linguagens. Com isso, para o TCC1 foi desenvolvido interfaces para as linguagens C (Seção 4.1) e AMPL (Seção 4.2) e, para o TCC2, foi desenvolvido interfaces para as linguagens Python (Seção 4.3) e R (Seção 4.4).

Torna-se evidente, portanto, que é interessante a criação de novas interfaces e, com isso, como trabalhos futuros, compreende-se o desenvolvimento de interoperabilidade com mais linguagens de programação ou de modelagem matemática que possam interessar as diversas áreas que utilizam programação não linear.

Referências

ALAHMADI, S. et al. Performance analysis of sparse matrix-vector multiplication (spmv) on graphics processing units (gpus). *Eletronics*, n. 10, 2020. ISSN 2079-9292. Citado na página 24.

ANDREANI, R. et al. On augmented lagrangian methods with general lower-level constraints. 2003. Citado na página 23.

BAPTISTA, E. C. Método da função lagrangiana aumentada-barreira logarítmica para a solução do problema de fluxo de potência ótimo. 2001. Citado na página 22.

BIRGIN, E. G.; MARTíNEZ, J. M. Complexity and performance of an Augmented Lagrangian algorithm. *Optimization Methods and Software*, v. 35, n. 5, p. 885–920, 2020. Citado na página 31.

BUENO, L. F.; SENNE, T. A.; OLIVEIRA, J. R. S. de. Investigando a eficiência de algencan quando combinado com o método de newton em problemas de empacotamento de círculo. 2019. Citado na página 22.

CAVASOTTI, J. P. H. Aplicação da programação não linear na otimização de um suporte de parede sob ação vertical de uma força. 2018. Citado na página 19.

COCIAN, L. F. E. *Manual da Linguagem C*. 1. ed. [S.l.: s.n.], 2004. ISBN 85-7528-107-0. Citado na página 39.

COURSERA. *What Is Python Used For? A Beginner's Guide*. 2022. Disponível em: <https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python>. Acesso em: 15/12/2022. Citado na página 45.

DANG, H.-V.; SCHMIDT, B. The sliced coo format for sparse matrix-vector multiplication on cuda-enabled gpus. *Procedia Computer Science*, v. 9, p. 57–66, 2012. ISSN 1877-0509. Citado na página 25.

FOURER DAVID M. GAY, B. W. K. R. *AMPL: A Modeling Language for Mathematical Programming*. 2. ed. [S.l.: s.n.], 2002. ISBN 0-534-38809-4. Citado na página 41.

GNU. *Interoperability with C*. 2004. Disponível em: <https://gcc.gnu.org/onlinedocs/gfortran/Interoperability-with-C.html>. Acesso em: 07/09/2022. Citado na página 26.

GNU. *ISO C BINDING*. 2004. Disponível em: <https://gcc.gnu.org/onlinedocs/gfortran/ISO_005fC_005fBINDING.html>. Acesso em: 17/09/2022. Citado na página 26.

LOBATO, R. D. Algoritmos para problemas de programação não-linear com variáveis inteiras e contínuas. 2009. Citado na página 22.

LUENBERGER, D. G.; YE, Y. *Linear and Nonlinear Programming*. [S.l.: s.n.], 1984. ISBN 978-0-387-74502-2. Citado na página 22.

MARTíNEZ, J. M. Otimização prática usando o lagrangiano aumentado. 2006. Citado na página 19.

ORENGO, G. *Fortran 95: curso básico*. 1. ed. [S.l.: s.n.], 2014. Citado na página 26.

R. *What is R?* 2022. Disponível em: <https://www.r-project.org/about.html>. Acesso em: 23/01/2023. Citado na página 50.

SAAD, Y. *Iterative Methods for Sparse Linear Systems*. 2. ed. [S.l.]: Society for Industrial and Applied Mathematics, 2003. ISBN 978-0-89871-534-7. Citado na página 23.

SANTOS, L. M. et al. Algoritmos de otimização aplicados à solução de sistemas estruturais não-lineares com restrições: uma abordagem utilizando os métodos da penalidade e do lagrangiano aumentado. v. 8, n. 3, p. 345–361, 2010. Citado 2 vezes nas páginas 21 e 22.

SILVA et al. Relação entre modelos de programação não-linear com incerteza no conjunto de restrições. v. 28, n. 3, p. 383–398, 2008. Citado na página 21.

SOUSA Álan Crístoffer e. Curso básico de python. 2020. Citado na página 45.

STOER, J.; BULIRSCH, R. *Introduction to Numerical Analysis*. 3. ed. [S.l.: s.n.], 2002. Citado 2 vezes nas páginas 23 e 24.

TEAM, R. C. R language definition. *Vienna, Austria: R foundation for statistical computing*, 2000. Citado na página 50.

TECHOPEDIA. *What Does Wrapper Mean?* 2018. Disponível em: <https://www.techopedia.com/definition/4389/wrapper-software-engineering>. Acesso em: 18/01/2023. Citado na página 41.

Anexos

ANEXO A – Definição das rotinas de avaliação em Fortran

Código 9 – Rotina de avaliação `evalf` em Fortran

```
subroutine evalf(n,x,f,inform,pdataptr)

  implicit none

  integer, intent(in) :: n
  integer, intent(inout) :: inform
  real(kind=8), intent(out) :: f
  type(c_ptr), optional, intent(in) :: pdataptr

  real(kind=8), intent(in) :: x(n)

  type(pdata_type), pointer :: pdata

  call c_f_pointer(pdataptr,pdata)
  pdata%counters(1) = pdata%counters(1) + 1

  f = (1.0d0 - x(1)) ** 2.0d0

end subroutine evalf
```

Código 10 – Rotina de avaliação `evalg` em Fortran

```
subroutine evalg(n,x,g,inform,pdataptr)

  implicit none

  integer, intent(in) :: n
  integer, intent(inout) :: inform
  type(c_ptr), optional, intent(in) :: pdataptr

  real(kind=8), intent(in) :: x(n)
  real(kind=8), intent(out) :: g(n)

  type(pdata_type), pointer :: pdata
```

```
13
14    call c_f_pointer(pdataptr,pdata)
15    pdata%counters(2) = pdata%counters(2) + 1
16
17    g(1) = -2.0d0 + 2.0d0 * x(1)
18    g(2) = 0
19
20  end subroutine evalg
```

Código 11 – Rotina de avaliação `evalc` em Fortran

```
1   subroutine evalc(n,x,m,p,c,inform,pdataptr)
2
3     implicit none
4
5     integer, intent(in) :: m,n,p
6     integer, intent(inout) :: inform
7     type(c_ptr), optional, intent(in) :: pdataptr
8
9     real(kind=8), intent(in) :: x(n)
10    real(kind=8), intent(out) :: c(m+p)
11
12    type(pdata_type), pointer :: pdata
13
14    call c_f_pointer(pdataptr,pdata)
15    pdata%counters(3) = pdata%counters(3) + 1
16
17    c(1) = 10.0d0 * x(2) - 10.0d0 * x(1) ** 2.0d0
18
19  end subroutine evalc
```

Código 12 – Rotina de avaliação `evalj` em Fortran

```
1   subroutine evalj(n,x,m,p,ind,sorted,jsta,jlen,lim,jvar,jval,
      inform,pdataptr)
2
3     implicit none
4
5     integer, intent(in) :: lim,m,n,p
6     integer, intent(inout) :: inform
7     type(c_ptr), optional, intent(in) :: pdataptr
8
```

```
  logical, intent(in) :: ind(m+p)
  real(kind=8), intent(in) :: x(n)
  logical, intent(out) :: sorted(m+p)
  integer, intent(out) :: jsta(m+p),jlen(m+p),jvar(lim)
  real(kind=8), intent(out) :: jval(lim)

  integer :: i
  type(pdata_type), pointer :: pdata

  call c_f_pointer(pdataptr,pdata)
  pdata%counters(4) = pdata%counters(4) + 1

  if ( ind(1) ) then
    if ( lim .lt. n ) then
      inform = -94
      return
    end if

    jsta(1) = 1
    jlen(1) = n

    jvar(1:n) = (/ (i,i=1,n) /)

    jval(1) = - 20.0d0 * x(1)
    jval(2) = 10.0d0

    sorted(1) = .true.
  end if

end subroutine evalj
```

Código 13 – Rotina de avaliação `evalhl` em Fortran

```
subroutine evalhl(n,x,m,p,lambda,lim,inclf,hlnnz,hlrow,hlcol,
   hlval,inform,pdataptr)

  implicit none

  logical, intent(in) :: inclf
  integer, intent(in) :: m,n,lim,p
  integer, intent(out) :: hlnnz
  integer, intent(inout) :: inform
```

```
9   type(c_ptr), optional, intent(in) :: pdataptr
10
11  real(kind=8), intent(in) :: lambda(m+p),x(n)
12  integer, intent(out) :: hlrow(lim),hlcol(lim)
13  real(kind=8), intent(out) :: hlval(lim)
14
15  type(pdata_type), pointer :: pdata
16
17  call c_f_pointer(pdataptr,pdata)
18  pdata%counters(5) = pdata%counters(5) + 1
19
20  hlnnz = 0
21
22  if ( inclf ) then
23    if ( hlnnz + 2 .gt. lim ) then
24      inform = -95
25      return
26    end if
27
28    hlnnz = hlnnz + 1
29
30    hlrow(hlnnz) = 1
31    hlcol(hlnnz) = 1
32    hlval(hlnnz) = 2.0d0
33
34  end if
35
36  if ( hlnnz + 1 .gt. lim ) then
37    inform = -95
38    return
39  end if
40
41  hlnnz = hlnnz + 1
42
43  hlrow(hlnnz) = 1
44  hlcol(hlnnz) = 1
45  hlval(hlnnz) = lambda(1) * (- 20.0d0)
46
47 end subroutine evalhl
```

ANEXO B – Definição das rotinas de avaliação em C

Código 14 – Rotina de avaliação `evalf` em C

```c
void evalf (int *n, double *x, double *f, int *ierr, pdata_
    type *pdata) {
  *f = pow(1.0 - x[0], 2.0 );
  pdata->counters[0]++;
}
```

Código 15 – Rotina de avaliação `evalg` em C

```c
void evalg (int *n, double *x, double *g, int *ierr, pdata_
    type *pdata) {
  g[0] = -2.0 + 2.0 * x[0];
  g[1] = 0;
  pdata->counters[1]++;
}
```

Código 16 – Rotina de avaliação `evalc` em C

```c
void evalc(int *n, double *x, int *m, int *p, double *c, int
    *ierr, pdata_type *pdata) {
  c[0] = 10.0 * x[1] - 10.0 * pow(x[0], 2.0);
  printf("c[0]: %lf\n\n", c[0]);
  pdata->counters[2]++;
}
```

Código 17 – Rotina de avaliação `evalj` em C

```c
void evalj(int *n, double *x, int *m, int *p, int *ind,
     int *jsorted, int *jsta, int *jlen, int *lim, int *jvar,
     double *jval, int *ierr, pdata_type *pdata) {
  if ( ind[0] ) {
    if ( *lim < *n ) {
      *ierr = -94;
      return;
    }

    jsta[0] = 1;
```

```
    jlen[0] = *n;

    for (int i = 0; i < *n; i++) jvar[i] = i+1;

    jval[0] = -20.0 * x[0];
    jval[1] = 10.0;

    jsorted[0] = 1;
  }

  pdata->counters[3]++;
}
```

Código 18 – Rotina de avaliação `evalhl` em C

```
void evalhl(int *n, double *x, int *m, int *p, double *lambda
   , int *lim,
       int *inclf, int *hlnnz, int *hlrow, int *hlcol, double
   *hlval,
       int *ierr, pdata_type *pdata) {

  *hlnnz = 0;

  if ( *inclf ) {
    if ( *hlnnz + 2 > *lim ) {
      *ierr = -95;
      return;
    }

    hlrow[*hlnnz] = 1;
    hlcol[*hlnnz] = 1;
    hlval[*hlnnz] = 2.0;
  }

  if ( *hlnnz + 1 > *lim ) {
    *ierr = -95;
    return;
  }

  *hlnnz = *hlnnz + 1;
  hlrow[*hlnnz] = 1;
  hlcol[*hlnnz] = 1;
```

```
26    hlval[*hlnnz] = lambda[0] * (-20.0);
27    *hlnnz = *hlnnz + 1;
28
29    pdata->counters[4]++;
30  }
```

ANEXO C – Descrição do problema de otimização para interface com AMPL

Código 19 – Descrição do problema de otimização em AMPL

```
var x1 >= 0, := - 1.2;
var x2 >= 0, := 1.0;

minimize f: (1 - x1) ^ 2;

s.t. g: 10 * (x2 - x1 ^ 2) = 0;
```

ANEXO D – Definição das rotinas de avaliação para interface com Python

Código 20 – Rotina de avaliação `evalf` em Python

```python
def evalf_py(*x):
  flag = 0
  f = (1.0 - x[0]) ** 2

  return f, flag
```

Código 21 – Rotina de avaliação `evalg` em Python

```python
def evalg_py(*x):
  flag = 0
  g = zeros(2)
  g[0] = -2 + 2 * x[0]
  g[1] = 0
  g = tuple(g)

  return g, flag
```

Código 22 – Rotina de avaliação `evalc` em Python

```python
def evalc_py(*x):
  flag = 0

  c = zeros(1)
  c[0] = 10 * x[1] - 10 * x[0] ** 2

  c = tuple(c)

  return c, flag
```

Código 23 – Rotina de avaliação `evalj` em Python

```python
def evalj_py(*tuple_c):
  # tuple_c = (n, x[0], x[1], ...)
  n = int(tuple_c[0])
  x = tuple_c[1:]
  flag = 0
```

```
6
7   jsta = zeros(1)
8   jlen = zeros(1)
9   jvar = zeros(n)
10  jval = zeros(2)
11  jsta[0] = 1
12  jlen[0] = n
13
14  for index in range(n):
15    jvar[index] = index + 1
16
17  jval[0] = -20 * x[0]
18  jval[1] = 10
19
20  jsta = tuple(jsta)
21  jlen = tuple(jlen)
22  jvar = tuple(jvar)
23  jval = tuple(jval)
24
25  return jsta, jlen, jvar, jval, flag
```

Código 24 – Rotina de avaliação `evalhl` em Python

```
1  def evalhl_py(*tuple_c):
2    # tuple_c = (n, lim, inclf, x[0], x[1], ..., _lambda[0], _
     lambda[1], ...)
3    n = int(tuple_c[0])
4    lim = tuple_c[1]
5    inclf = tuple_c[2]
6    x = tuple_c[3:n+3]
7    _lambda = tuple_c[n+3:]
8    flag = 0
9    hlnnz = 0
10   ierr = 0
11
12   hlrow = zeros(2)
13   hlcol = zeros(2)
14   hlval = zeros(2)
15
16   if inclf:
17     if hlnnz + 2 > lim:
18       ierr = -95
```

```
      hlrow = tuple(hlrow)
      hlcol = tuple(hlcol)
      hlval = tuple(hlval)
      return hlrow, hlcol, hlval, hlnnz, ierr, flag

    hlrow[0] = 1
    hlcol[0] = 1
    hlval[0] = 2

  if hlnnz + 1 > lim:
    ierr = -95
    hlrow = tuple(hlrow)
    hlcol = tuple(hlcol)
    hlval = tuple(hlval)
    return hlrow, hlcol, hlval, hlnnz, ierr, flag

  hlnnz = hlnnz + 1
  hlrow[1] = 1
  hlcol[1] = 1
  hlval[1] = _lambda[0] * (-20)

  hlnnz = hlnnz + 1

  hlrow = tuple(hlrow)
  hlcol = tuple(hlcol)
  hlval = tuple(hlval)

  return hlrow, hlcol, hlval, hlnnz, ierr, flag
```

ANEXO E – Definição das rotinas de avaliação para interface com R

Código 25 – Rotina de avaliação `evalf` em R

```r
evalf_r <- function(x) {
  flag <- 0L
  f <- (1.0 - x[1]) ** 2

  return(list(f, flag))
}
```

Código 26 – Rotina de avaliação `evalg` em R

```r
evalg_r <- function(x) {
  flag <- 0L
  g <- vector()
  g[1] <- -2.0 + 2.0 * x[1]
  g[2] <- 0

  return(list(g, flag))
}
```

Código 27 – Rotina de avaliação `evalc` em R

```r
evalc_r <- function(x) {
  flag <- 0L
  c <- vector()
  c[1] <- 10 * x[2] - 10 * x[1] ** 2

  return(list(c, flag))
}
```

Código 28 – Rotina de avaliação `evalj` em R

```r
evalj_r <- function(x, n) {
  flag <- 0L
  jsta <- integer()
  jlen <- integer()
  jvar <- integer()
  jval <- vector()
```

```
  jsta[1] <- 1L
  jlen[1] <- n

  index <- 0L
  while (index < n) {
    jvar[index + 1L] <- index + 1L
    index <- index + 1L
  }

  jval[1] <- -20.0 * x[1]
  jval[2] <- 10.0

  return(list(jsta, jlen, jvar, jval, flag))
}
```

Código 29 – Rotina de avaliação `evalhl` em R

```
evalhl_r <- function(x, n, lim, inclf, lambda) {
  flag <- 0L
  hlnnz <- 0L
  ierr <- 0L
  hlrow <- integer()
  hlcol <- integer()
  hlval <- vector()

  if (inclf) {
    if (hlnnz + 2 > lim) {
      ierr <- -95L
      return(list(hlrow, hlcol, hlval, hlnnz, ierr, flag))
    }

    hlrow[1] <- 1L
    hlcol[1] <- 1L
    hlval[1] <- 2
  }

  if (hlnnz + 1 > lim) {
    ierr <- -95L
    return(list(hlrow, hlcol, hlval, hlnnz, ierr, flag))
  }

  hlnnz <- hlnnz + 1L
```

```
  hlrow[2] <- 1L
  hlcol[2] <- 1L
  hlval[2] <- lambda[1] * (-20)
  hlnnz <- hlnnz + 1L

  return(list(hlrow, hlcol, hlval, hlnnz, ierr, flag))
}
```

ANEXO F – Definição dos *wrappers* para as rotinas de avaliação em AMPL

Código 30 – *Wrapper* para rotina de avaliação `evalf` em AMPL

```
void evalf(int *n, double *x, double *f, int *ierr, pdata_
    type *pdata) {
  int *flag = 0;
  *f = objval(0, (real *)x, (fint *)flag);
  pdata->counters[0]++;
}
```

Código 31 – *Wrapper* para rotina de avaliação `evalg` em AMPL

```
void evalg(int *n, double *x, double *g, int *ierr, pdata_
    type *pdata) {
  int *flag = 0;
  objgrd(0, (real *)x, (real *)g, (fint *)flag);
  pdata->counters[1]++;
}
```

Código 32 – *Wrapper* para rotina de avaliação `evalc` em AMPL

```
void evalc(int *n, double *x, int *m, int *p, double *c, int
    *ierr, pdata_type *pdata) {
  int *flag = 0;
  double cl, cu;

  conval((real *)x, (real *)campl, (fint *)flag);

  for (int i = 0; i < n_con; i++) {
    cl = LUrhs[2 * i];
    cu = LUrhs[2 * i + 1];

    if (isBreakingConstraint(cl, cu)) {
      c[cmap[i]] = cl - campl[i];
      c[cmap[i]+1] = campl[i] - cu;
    }
    else {
      if (cu < Infinity) c[cmap[i]] = campl[i] - cu;
```

```
      else c[cmap[i]] = cl - campl[i];
    }
  }

  pdata->counters[2]++;
}
```

Código 33 – *Wrapper* para rotina de avaliação `evalj` em AMPL

```
void evalj(int *n, double *x, int *m, int *p, int *ind,
     int *jsorted, int *jsta, int *jlen, int *lim, int *jvar,
     double *jval, int *ierr, pdata_type *pdata) {
  int flag = 0, nnz = 0;
  double cl, cu;

  for (int i = 0; i < *n; i++) jlen[i] = 0;

  double *cg = malloc (*n * sizeof (double));

  for (int i = 0; i < n_con; i++) {
    if (ind[cmap[i]]) {
      congrd (i, (real *)x, (real *)cg, (fint *)&flag);

      cl = LUrhs[2 * i];
      cu = LUrhs[2 * i + 1];

      jsta[cmap[i]] = nnz+1;

      for (int j = 0; j < *n; j++) {
  if (fabs (cg[j]) > DBL_EPSILON) {
    jvar[nnz] = j+1;
    jval[nnz] = cg[j];
    nnz++;
    jlen[cmap[i]]++;
  }
      }

      jsorted[cmap[i]] = 1;

      if (isBreakingConstraint(cl, cu) && ind[cmap[i]+1]) {
  jsta[cmap[i]+1] = nnz+1;

```

```
34    for (int j = 0; j < *n; j++) {
35       if (fabs (cg[j]) > DBL_EPSILON) {
36        jvar[nnz] = j+1;
37        jval[nnz] = cg[j];
38        nnz++;
39        jlen[cmap[i]+1]++;
40       }
41    }
42
43    jsorted[cmap[i]+1] = 1;
44        }
45      }
46    }
47
48    pdata->counters[3]++;
49    free (cg);
50  }
```

Código 34 – *Wrapper* para rotina de avaliação `evalhl` em AMPL

```
1  void evalhl(int *n, double *x, int *m, int *p, double *lambda
       , int *lim,
2          int *inclf, int *hlnnz, int *hlrow, int *hlcol, double
       *hlval,
3          int *ierr, pdata_type *pdata) {
4    *hlnnz = 0;
5    int k;
6    double *OW = NULL, *coeff = NULL;
7
8    if (*inclf) {
9      OW = malloc (n_obj * sizeof (double));
10     for (int i = 0; i < n_obj; i++) OW[i] = 1.0;
11   }
12
13   coeff = malloc (n_con * sizeof (double));
14
15   for (int i = 0; i < n_con; i++) {
16     coeff[i] = lambda[cmap[i]];
17   }
18
19   sphes (hlval, -1, OW, coeff);
20
```

```
21   for ( int j = 0; j < *n; j++ ) {
22     for ( int i = sputinfo->hcolstarts[j]; i < sputinfo->
     hcolstarts[j+1]; i++ ) {
23       hlrow[i] = sputinfo->hrownos[i] + 1;
24       hlcol[i] = j + 1;
25     }
26   }
27
28   *hlnnz = sputinfo->hcolstarts[*n];
29   pdata->counters[4]++;
30
31   if (*inclf) free (OW);
32
33   free (coeff);
34 }
```

ANEXO G – Definição dos *wrappers* para as rotinas de avaliação em Python

Código 35 – *Wrapper* para rotina de avaliação `evalf` em Python

```
void evalf(int *n, double *x, double *f, int *ierr, pdata_
    type *pdata) {
  PyObject *evalfName, *evalfModule, *evalfDict,
           *evalfFunc, *evalfResult, *tuple;

  evalfName = PyUnicode_FromString((char*)"problem");
  evalfModule = PyImport_Import(evalfName);
  evalfDict = PyModule_GetDict(evalfModule);
  evalfFunc = PyDict_GetItemString(evalfDict, (char*)funcs.
    evalf_name);

  if (PyCallable_Check(evalfFunc)) {
    tuple = PyTuple_New(*n);

    for (int i = 0; i < *n; i++) {
      PyTuple_SetItem(tuple, i, PyFloat_FromDouble(x[i]));
      PyErr_Print();
    }

    evalfResult = PyObject_CallObject(evalfFunc, tuple);
    PyErr_Print();
  } else {
    PyErr_Print();
  }

  *f = PyFloat_AsDouble(PyTuple_GetItem(evalfResult, 0));
  pdata->counters[0]++;
}
```

Código 36 – *Wrapper* para rotina de avaliação `evalg` em Python

```
void evalg(int *n, double *x, double *g, int *ierr, pdata_
    type *pdata) {
  PyObject *evalgName, *evalgModule, *evalgDict,
```

```
            *evalgFunc, *evalgResult, *tuple, *g_result;

  evalgName = PyUnicode_FromString((char*)"problem");
  evalgModule = PyImport_Import(evalgName);
  evalgDict = PyModule_GetDict(evalgModule);
  evalgFunc = PyDict_GetItemString(evalgDict, (char*)funcs.
   evalg_name);

  if (PyCallable_Check(evalgFunc)) {
    tuple = PyTuple_New(*n);

    for (int i = 0; i < *n; i++) {
      PyTuple_SetItem(tuple, i, PyFloat_FromDouble(x[i]));
      PyErr_Print();
    }

    evalgResult = PyObject_CallObject(evalgFunc, tuple);
    PyErr_Print();
  } else {
    PyErr_Print();
  }

  g_result = PyTuple_GetItem(evalgResult, 0);
  PyErr_Print();

  if (PyTuple_Check(g_result)) {
    for (int i = 0; i < PyTuple_Size(g_result); i++) {
      g[i] = PyFloat_AsDouble(PyTuple_GetItem(g_result, i));
      PyErr_Print();
    }
  }

  pdata->counters[1]++;
}
```

Código 37 – *Wrapper* para rotina de avaliação `evalc` em Python

```
void evalc(int *n, double *x, int *m, int *p, double *c, int
   *ierr, pdata_type *pdata) {
  PyObject *evalcName, *evalcModule, *evalcDict,
           *evalcFunc, *evalcResult, *tuple, *c_result;

```

```
  evalcName = PyUnicode_FromString((char*)"problem");
  evalcModule = PyImport_Import(evalcName);
  evalcDict = PyModule_GetDict(evalcModule);
  evalcFunc = PyDict_GetItemString(evalcDict, (char*)funcs.
   evalc_name);

  if (PyCallable_Check(evalcFunc)) {
    tuple = PyTuple_New(*n);

    for (int i = 0; i < *n; i++) {
      PyTuple_SetItem(tuple, i, PyFloat_FromDouble(x[i]));
      PyErr_Print();
    }

    evalcResult = PyObject_CallObject(evalcFunc, tuple);
    PyErr_Print();
  } else {
    PyErr_Print();
  }

  c_result = PyTuple_GetItem(evalcResult, 0);
  PyErr_Print();

  if (PyTuple_Check(c_result)) {
    for (Py_ssize_t i = 0; i < PyTuple_Size(c_result); i++) {
      c[i] = PyFloat_AsDouble(PyTuple_GetItem(c_result, i));
      PyErr_Print();
    }
  }

  pdata->counters[2]++;
}
```

Código 38 – *Wrapper* para rotina de avaliação `evalj` em Python

```
void evalj (int *n, double *x, int *m, int *p, int *ind,
            int *jsorted, int *jsta, int *jlen, int *lim, int
   *jvar,
            double *jval, int *ierr, pdata_type *pdata) {
  PyObject *evaljName, *evaljModule, *evaljDict,
           *evaljFunc, *evaljResult, *tuple, *jsta_result,
           *jlen_result, *jvar_result, *jval_result;
```

```
7
8   evaljName = PyUnicode_FromString((char*)"problem");
9   evaljModule = PyImport_Import(evaljName);
10  evaljDict = PyModule_GetDict(evaljModule);
11  evaljFunc = PyDict_GetItemString(evaljDict, (char*)funcs.
     evalj_name);
12
13  if (ind[0]) {
14    if (*lim < *n) {
15      *ierr = -94;
16      return;
17    }
18
19    if (PyCallable_Check(evaljFunc)) {
20      tuple = PyTuple_New(*n + 1);
21      PyErr_Print();
22      PyTuple_SetItem(tuple, 0, PyFloat_FromDouble((double)*n
    ));
23      PyErr_Print();
24
25      for (int i = 1, j = 0; i < *n + 1; i++, j++) {
26        PyTuple_SetItem(tuple, i, PyFloat_FromDouble(x[j]));
27        PyErr_Print();
28      }
29
30      evaljResult = PyObject_CallObject(evaljFunc, tuple);
31      PyErr_Print();
32    } else {
33      PyErr_Print();
34    }
35
36    jsta_result = PyTuple_GetItem(evaljResult, 0);
37    PyErr_Print();
38
39    if (PyTuple_Check(jsta_result)) {
40      for (int i = 0; i < PyTuple_Size(jsta_result); i++) {
41        jsta[i] = PyFloat_AsDouble(PyTuple_GetItem(jsta_
    result, i));
42        PyErr_Print();
43      }
44    }
```

```
45
46      jlen_result = PyTuple_GetItem(evaljResult, 1);
47      PyErr_Print();
48
49      if (PyTuple_Check(jlen_result)) {
50        for (int i = 0; i < PyTuple_Size(jlen_result); i++) {
51          jlen[i] = PyFloat_AsDouble(PyTuple_GetItem(jlen_
    result, i));
52          PyErr_Print();
53        }
54      }
55
56      jvar_result = PyTuple_GetItem(evaljResult, 2);
57      PyErr_Print();
58
59      if (PyTuple_Check(jvar_result)) {
60        for (int i = 0; i < PyTuple_Size(jvar_result); i++) {
61          jvar[i] = PyFloat_AsDouble(PyTuple_GetItem(jvar_
    result, i));
62          PyErr_Print();
63        }
64      }
65
66      jval_result = PyTuple_GetItem(evaljResult, 3);
67      PyErr_Print();
68
69      if (PyTuple_Check(jval_result)) {
70        for (int i = 0; i < PyTuple_Size(jval_result); i++) {
71          jval[i] = PyFloat_AsDouble(PyTuple_GetItem(jval_
    result, i));
72          PyErr_Print();
73        }
74      }
75
76      jsorted[0] = 1;
77    }
78
79    pdata->counters[3]++;
80  }
```

Código 39 – *Wrapper* para rotina de avaliação `evalhl` em Python

```
void evalhl (int *n, double *x, int *m, int *p, double *
   lambda, int *lim,
2             int *inclf, int *hlnnz, int *hlrow, int *hlcol,
   double *hlval,
3             int *ierr, pdata_type *pdata) {
4  PyObject *evalhlName, *evalhlModule, *evalhlDict,
5           *evalhlFunc, *evalhlResult, *tuple, *hlrow_result,
6           *hlcol_result, *hlval_result;
7
8  evalhlName = PyUnicode_FromString((char*)"problem");
9  evalhlModule = PyImport_Import(evalhlName);
10 evalhlDict = PyModule_GetDict(evalhlModule);
11 evalhlFunc = PyDict_GetItemString(evalhlDict, (char*)funcs.
   evalhl_name);
12
13 if (PyCallable_Check(evalhlFunc)) {
14   int tuple_lenght = (*n) + (*m) + (*p) + 3;
15   tuple = PyTuple_New(tuple_lenght);
16   PyErr_Print();
17
18   PyTuple_SetItem(tuple, 0, PyFloat_FromDouble((double)*n))
   ;
19   PyErr_Print();
20
21   PyTuple_SetItem(tuple, 1, PyFloat_FromDouble(*lim));
22   PyErr_Print();
23
24   PyTuple_SetItem(tuple, 2, PyFloat_FromDouble(*inclf));
25   PyErr_Print();
26
27   for (int i = 3, j = 0; i < *n + 3; i++, j++) {
28     PyTuple_SetItem(tuple, i, PyFloat_FromDouble(x[j]));
29     PyErr_Print();
30   }
31
32   for (int i = *n + 3, j = 0; i < tuple_lenght; i++, j++) {
33     PyTuple_SetItem(tuple, i, PyFloat_FromDouble(lambda[j])
   );
34     PyErr_Print();
```

```
35      }
36
37      evalhlResult = PyObject_CallObject(evalhlFunc, tuple);
38      PyErr_Print();
39      *ierr = PyFloat_AsDouble(PyTuple_GetItem(evalhlResult, 4)
    );
40      PyErr_Print();
41
42      if (*ierr == -95) {
43        return;
44      }
45    } else {
46      PyErr_Print();
47    }
48
49    hlrow_result = PyTuple_GetItem(evalhlResult, 0);
50    PyErr_Print();
51
52    if (PyTuple_Check(hlrow_result)) {
53      for (int i = 0; i < PyTuple_Size(hlrow_result); i++) {
54        hlrow[i] = (int)PyFloat_AsDouble(PyTuple_GetItem(hlrow_
    result, i));
55        PyErr_Print();
56      }
57    }
58
59    hlcol_result = PyTuple_GetItem(evalhlResult, 1);
60    PyErr_Print();
61
62    if (PyTuple_Check(hlcol_result)) {
63      for (int i = 0; i < PyTuple_Size(hlcol_result); i++) {
64        hlcol[i] = (int)PyFloat_AsDouble(PyTuple_GetItem(hlcol_
    result, i));
65        PyErr_Print();
66      }
67    }
68
69    hlval_result = PyTuple_GetItem(evalhlResult, 2);
70    PyErr_Print();
71
72    if (PyTuple_Check(hlval_result)) {
```

```
73      for (int i = 0; i < PyTuple_Size(hlval_result); i++) {
74        hlval[i] = PyFloat_AsDouble(PyTuple_GetItem(hlval_
     result, i));
75        PyErr_Print();
76      }
77    }
78
79    *hlnnz = (int)PyFloat_AsDouble(PyTuple_GetItem(evalhlResult
     , 3));
80    PyErr_Print();
81
82    pdata->counters[4]++;
83  }
```

ANEXO H – Definição dos *wrappers* para as rotinas de avaliação em R

Código 40 – *Wrapper* para rotina de avaliação `evalf` em R

```
void evalf(int *n, double *x, double *f, int *ierr, pdata_
   type *pdata) {
  SEXP arg_x, evalf_call, ret;
  double **evalf_result_real;
  int **evalf_result_integer, errorOccurred;

  Rf_protect(arg_x = Rf_allocVector(REALSXP, *n));
  memcpy(REAL(arg_x), x, *n * sizeof(double));
  Rf_protect(evalf_call = Rf_lang2(Rf_install((char*)funcs.
   evalf_name), arg_x));
  ret = R_tryEval(evalf_call, R_GlobalEnv, &errorOccurred);

  if (!errorOccurred) {
    evalf_result_real = malloc(sizeof(double*));
    evalf_result_integer = malloc(sizeof(double*));

    for (int i = 0, j = 0, k = 0; i < LENGTH(ret); i++) {
      if (Rf_isReal(VECTOR_ELT(ret, i))) {
        evalf_result_real[j] = REAL(VECTOR_ELT(ret, i));
        j++;
      } else if (Rf_isInteger(VECTOR_ELT(ret, i))) {
        evalf_result_integer[k] = INTEGER(VECTOR_ELT(ret, i))
   ;
        k++;
      }
    }

    *f = evalf_result_real[0][0];
    pdata->counters[0]++;
  }
  else {
    printf("Error occurred calling R\n");
  }
```

```
31
32   free(evalf_result_real);
33   free(evalf_result_integer);
34   Rf_unprotect(2);
35 }
```

Código 41 – *Wrapper* para rotina de avaliação `evalg` em R

```
1  void evalg(int *n, double *x, double *g, int *ierr, pdata_
       type *pdata) {
2    SEXP arg_x, evalg_call, ret;
3    double **evalg_result_real;
4    int **evalg_result_integer, errorOccurred, g_lenght;
5
6    Rf_protect(arg_x = Rf_allocVector(REALSXP, *n));
7    memcpy(REAL(arg_x), x, *n * sizeof(double));
8    Rf_protect(evalg_call = Rf_lang2(Rf_install((char*)funcs.
       evalg_name), arg_x));
9    ret = R_tryEval(evalg_call, R_GlobalEnv, &errorOccurred);
10
11   if (!errorOccurred) {
12     evalg_result_real = malloc(sizeof(double*));
13     evalg_result_integer = malloc(sizeof(double*));
14
15     for (int i = 0, j = 0, k = 0; i < LENGTH(ret); i++) {
16       if (Rf_isReal(VECTOR_ELT(ret, i))) {
17         evalg_result_real[j] = REAL(VECTOR_ELT(ret, i));
18         j++;
19       } else if (Rf_isInteger(VECTOR_ELT(ret, i))) {
20         evalg_result_integer[k] = INTEGER(VECTOR_ELT(ret, i))
       ;
21         k++;
22       }
23     }
24
25     g_lenght = Rf_xlength(VECTOR_ELT(ret, 0));
26
27     for (int i = 0; i < g_lenght; i++) {
28       g[i] = evalg_result_real[0][i];
29     }
30
31     pdata->counters[1]++;
```

```
  }
  else {
    printf("Error occurred calling R\n");
  }

  free(evalg_result_real);
  free(evalg_result_integer);
  Rf_unprotect(2);
}
```

Código 42 – *Wrapper* para rotina de avaliação `evalc` em R

```
void evalc(int *n, double *x, int *m, int *p, double *c, int
   *ierr, pdata_type *pdata) {
  SEXP arg_x, evalc_call, ret;
  double **evalc_result_real;
  int **evalc_result_integer, errorOccurred, c_lenght;

  Rf_protect(arg_x = Rf_allocVector(REALSXP, *n));
  memcpy(REAL(arg_x), x, *n * sizeof(double));
  Rf_protect(evalc_call = Rf_lang2(Rf_install((char*)funcs.
   evalc_name), arg_x));
  ret = R_tryEval(evalc_call, R_GlobalEnv, &errorOccurred);

  if (!errorOccurred) {
    evalc_result_real = malloc(sizeof(double*));
    evalc_result_integer = malloc(sizeof(double*));

    for (int i = 0, j = 0, k = 0; i < LENGTH(ret); i++) {
      if (Rf_isReal(VECTOR_ELT(ret, i))) {
        evalc_result_real[j] = REAL(VECTOR_ELT(ret, i));
        j++;
      } else if (Rf_isInteger(VECTOR_ELT(ret, i))) {
        evalc_result_integer[k] = INTEGER(VECTOR_ELT(ret, i))
   ;
        k++;
      }
    }

    c_lenght = Rf_xlength(VECTOR_ELT(ret, 0));

    for (int i = 0; i < c_lenght; i++) {
```

```
28        c[i] = evalc_result_real[0][i];
29      }
30
31      pdata->counters[2]++;
32    }
33    else {
34      printf("Error occurred calling R\n");
35    }
36
37    free(evalc_result_real);
38    free(evalc_result_integer);
39    Rf_unprotect(2);
40  }
```

Código 43 – *Wrapper* para rotina de avaliação `evalj` em R

```
1  void evalj(int *n, double *x, int *m, int *p, int *ind,
2             int *jsorted, int *jsta, int *jlen, int *lim,
   int *jvar,
3             double *jval, int *ierr, pdata_type *pdata) {
4    SEXP arg_x, arg_n, evalj_call, ret;
5    double **evalj_result_real;
6    int **evalj_result_integer, errorOccurred, jsta_lenght,
   jlen_lenght, jvar_lenght, jval_lenght;
7
8    Rf_protect(arg_x = Rf_allocVector(REALSXP, *n));
9    memcpy(REAL(arg_x), x, *n * sizeof(double));
10   Rf_protect(arg_n = Rf_ScalarInteger(*n));
11   Rf_protect(evalj_call = Rf_lang3(Rf_install((char*)funcs.
   evalj_name), arg_x, arg_n));
12
13   if (ind[0]) {
14     if (*lim < *n) {
15       *ierr = -94;
16       return;
17     }
18
19     ret = R_tryEval(evalj_call, R_GlobalEnv, &errorOccurred);
20
21     if (!errorOccurred) {
22       evalj_result_real = malloc(sizeof(double*));
23       evalj_result_integer = malloc(4 * sizeof(int*));
```

```
24
25      if (evalj_result_real == NULL || evalj_result_integer
    == NULL) {
26        printf("Allocation error.\n");
27        return;
28      }
29
30      for (int i = 0, j = 0, k = 0; i < LENGTH(ret); i++) {
31        if (Rf_isReal(VECTOR_ELT(ret, i))) {
32          evalj_result_real[j] = REAL(VECTOR_ELT(ret, i));
33          j++;
34        } else if (Rf_isInteger(VECTOR_ELT(ret, i))) {
35          evalj_result_integer[k] = INTEGER(VECTOR_ELT(ret, i
    ));
36          k++;
37        }
38      }
39
40      jsta_lenght = Rf_xlength(VECTOR_ELT(ret, 0));
41      for (int i = 0; i < jsta_lenght; i++) {
42        jsta[i] = evalj_result_integer[0][i];
43      }
44
45      jlen_lenght = Rf_xlength(VECTOR_ELT(ret, 1));
46      for (int i = 0; i < jlen_lenght; i++) {
47        jlen[i] = evalj_result_integer[1][i];
48      }
49
50      jvar_lenght = Rf_xlength(VECTOR_ELT(ret, 2));
51      for (int i = 0; i < jvar_lenght; i++) {
52        jvar[i] = evalj_result_integer[2][i];
53      }
54
55      jval_lenght = Rf_xlength(VECTOR_ELT(ret, 3));
56      for (int i = 0; i < jval_lenght; i++) {
57        jval[i] = evalj_result_real[0][i];
58      }
59
60      jsorted[0] = 1;
61      pdata->counters[3]++;
62    }
```

```
63   }
64   else {
65     printf("Error occurred calling R\n");
66   }
67
68   free(evalj_result_real);
69   free(evalj_result_integer);
70   Rf_unprotect(3);
71 }
```

Código 44 – *Wrapper* para rotina de avaliação `evalhl` em R

```
1  void evalhl(int *n, double *x, int *m, int *p, double *lambda
       , int *lim,
2              int *inclf, int *hlnnz, int *hlrow, int *hlcol,
        double *hlval,
3              int *ierr, pdata_type *pdata) {
4    SEXP arg_x, arg_n, arg_lim, arg_inclf, arg_lambda, evalhl_
       call, ret;
5    double **evalhl_result_real;
6    int **evalhl_result_integer, errorOccurred, hlrow_lenght,
       hlcol_lenght, hlval_lenght;
7
8    Rf_protect(arg_x = Rf_allocVector(REALSXP, *n));
9    memcpy(REAL(arg_x), x, *n * sizeof(double));
10   Rf_protect(arg_n = Rf_ScalarInteger(*n));
11   Rf_protect(arg_lim = Rf_ScalarInteger(*lim));
12   Rf_protect(arg_inclf = Rf_ScalarInteger(*inclf));
13   Rf_protect(arg_lambda = Rf_allocVector(REALSXP, *m + *p));
14   memcpy(REAL(arg_lambda), lambda, (*m + *p) * sizeof(double)
       );
15
16   Rf_protect(evalhl_call = Rf_lang6(Rf_install((char*)funcs.
       evalhl_name), arg_x, arg_n, arg_lim, arg_inclf, arg_lambda
       ));
17   ret = R_tryEval(evalhl_call, R_GlobalEnv, &errorOccurred);
18
19   if (!errorOccurred) {
20     evalhl_result_real = malloc(sizeof(double*));
21     evalhl_result_integer = malloc(5 * sizeof(double*));
22
```

```
23    if (evalhl_result_real == NULL || evalhl_result_integer
  == NULL) {
24      printf("Allocation error.\n");
25      return;
26    }
27
28    for (int i = 0, j = 0, k = 0; i < LENGTH(ret); i++) {
29      if (Rf_isReal(VECTOR_ELT(ret, i))) {
30        evalhl_result_real[j] = REAL(VECTOR_ELT(ret, i));
31        j++;
32      } else if (Rf_isInteger(VECTOR_ELT(ret, i))) {
33        evalhl_result_integer[k] = INTEGER(VECTOR_ELT(ret, i)
  );
34        k++;
35      }
36    }
37
38    hlrow_lenght = Rf_xlength(VECTOR_ELT(ret, 0));
39    for (int i = 0; i < hlrow_lenght; i++) {
40      hlrow[i] = evalhl_result_integer[0][i];
41    }
42
43    hlcol_lenght = Rf_xlength(VECTOR_ELT(ret, 1));
44    for (int i = 0; i < hlcol_lenght; i++) {
45      hlcol[i] = evalhl_result_integer[1][i];
46    }
47
48    hlval_lenght = Rf_xlength(VECTOR_ELT(ret, 2));
49    for (int i = 0; i < hlval_lenght; i++) {
50      hlval[i] = evalhl_result_real[0][i];
51    }
52
53    *hlnnz = evalhl_result_integer[2][0];
54    *ierr = evalhl_result_integer[3][0];
55
56    if (*ierr == -95) {
57      return;
58    }
59
60    pdata->counters[4]++;
61  }
```

```
62   else {
63     printf("Error occurred calling R\n");
64   }
65
66   free(evalhl_result_real);
67   free(evalhl_result_integer);
68   Rf_unprotect(6);
69 }
```