Universidade Estadual de Campinas
Instituto de Matemática Estatística e Computação Científica
Departamento de Matemática Aplicada

# Otimização via Internet

**Ricardo de Oliveira Lopes Jr**
Mestrado em Matemática Aplicada

Orientador: **Prof. Dr. Francisco A. M. Gomes Neto**

Campinas
Setembro de 2010

# Otimização via Internet

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por **Ricardo de Oliveira Lopes Jr** e aprovada pela comissão julgadora.

Campinas, 04 de agosto de 2010.

Prof. Dr. **Francisco A. M. Gomes Neto**
Orientador

**Banca examinadora:**
Prof. Dr. Francisco A. M. Gomes Neto (IMECC/UNICAMP)
Prof. Dr. Aurélio Ribeiro Leite de Oliveira (IMECC/UNICAMP)
Prof. Dr. Ernesto Julián Goldberg Birgin (IME/USP)

Dissertação apresentada ao Instituto de Matemática Estatística e Computação Científica, UNICAMP, como requisito parcial para a obtenção do título de **Mestre em Matemática Aplicada**.

**FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP**
Bibliotecária: Márcia AP. Pillon D'aloia CRB 8/5180

Lopes Junior, Ricardo de Oliveira
L881o Otimização via internet/Ricardo de Oliveira Lopes Junior-- Campinas, [S.P. : s.n.], 2010.

Orientador : Francisco de Assis Magalhães Gomes Neto
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Otimização. 2. Programação não-linear. 3. Internet. I. Gomes Neto, Francisco de Assis Magalhães. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

Título em inglês: Optimization over the internet

Palavras-chave em inglês (Keywords): 1. Optimization. 2. Nonlinear programming. 3. Internet.

Área de concentração: Matemática Aplicada

Titulação: Mestre em Matemática Aplicada

Banca examinadora: Prof. Dr. Francisco de Assis Magalhães Gomes Neto (IMEC-UNICAMP)
Prof. Dr. Ernesto Julián Goldberg Birgin (IME-USP)
Prof. Dr. Aurélio Ribeiro Leite de Oliveira (IMECC-UNICAMP)

Data da defesa: 04/08/2010

Programa de Pós-Graduação: Mestrado em Matemática Aplicada

**Dissertação de Mestrado defendida em 04 de agosto de 2010 e aprovada**

**Pela Banca Examinadora composta pelos Profs. Drs.**

**Prof.(a). Dr(a). FRANCISCO DE ASSIS MAGALHÃES GOMES NETO**

**Prof. (a). Dr (a). AURELIO RIBEIRO LEITE DE OLIVEIRA**

**Prof. (a). Dr (a). ERNESTO JULIÁN GOLDBERG BIRGIN**

*À minha família.*

# Agradecimentos

Ao meu orientador, Chico, pela confiança, ajuda e paciência infinita.

Aos professores do IMECC que contribuiram fortemente para a minha formação, entre eles, Aurélio, Cheti, Laércio, Moretti, Sandra e Vera.

Ao meu grande amigo Gabriel, por todas as discussões, caronas, projetos compartilhados e, principalmente, amizade, desde a graduação.

Aos nobres amigos Bruno, Moiséis e Elias, grandes figuras humanas, pela amizade e estadia na Morféia nos últimos anos.

Ao Vilamiu, até a data, último amigo remanescente do Predinho, pela ajuda pré e pós defesa.

À CAPES, via programa PROEX, e ao DMA/IMECC, pelo indispensável financiamento no início do projeto.

Aos meus pais, Ricardo e Gisela, pelo grandioso esforço despendido na minha formação.

Ao meu irmão Rodolfo, pelos ensinamentos em Economia.

À minha namorada, Amanda, pela compreensão e apoio, fundamentais para a conclusão deste trabalho.

# Resumo

Este trabalho apresenta um sistema remoto de resolução de problemas de otimização através da internet. O usuário visita a página do sistema, escolhe o programa de otimização que deseja utilizar, submete o seu problema e recebe a solução na tela ou por correio eletrônico. O objetivo do sistema é popularizar as bibliotecas computacionais desenvolvidas no país, assim como atender à demanda de pesquisadores e empresas por programas de otimização. No momento, estão disponíveis os pacotes GENCAN e ALGENCAN. O sistema permite que se use derivadas automáticas, geradas pelo pacote Adol-C, caso o programa exija o uso de derivadas e o usuário não possa fornecê-las. Uma fila de compilação e uma fila de processamento controlam o fluxo de submissões de problemas. O usuário pode visualizar essas filas e cancelar um trabalho submetido.

**Palavras chave:** otimização; programação não linear; internet

# Abstract

This work presents a remote system for solving optimization problems over the internet. The user visits the system website, chooses the optimization solver and submits a problem. The solution may be visualized in the browser window or by email. The objective of the system is to popularize the solvers developed by Brazilian researchers, as well as to meet the demand of researchers and companies for optimization programs. So far, the packages GENCAN and ALGENCAN are available. If the solver needs the derivatives of the functions and the user is not able to supply them, the system permits the use of automatic diferentiation, generated by the Adol-C package. A compilation queue and an execution queue control the server submission flow. The user may visualize these queues and cancel a submitted job.

**Key words:** optimization; nonlinear programming; internet

# Sumário

# Introdução

A otimização é um interessante ramo da matemática aplicada, dada a sua aplicação à solução de problemas ligados à produção, à engenharia, à economia e às ciências naturais.

Na UNICAMP, um grupo relativamente grande de professores tem se dedicado ao desenvolvimento teórico e à implantação computacional de programas voltados à solução de problemas de otimização. O mesmo acontece em outras universidades do país, como a USP e a UFRJ. Entretanto, a maioria desses programas não está facilmente disponível para os pesquisadores e as empresas que poderiam utilizá-los.

Alguns dos programas simplesmente não possuem uma página na internet, de forma que sua visibilidade é quase nula. Outros, embora disponíveis na *web*, não possuem uma interface simples, desestimulando o seu uso. Somando-se a isso o fato de que muitos dos potenciais usuários temem utilizar programas acadêmicos na solução de problemas reais sem antes testá-los com modelos pequenos, chega-se aos motivos principais para o baixo uso dos pacotes de otimização desenvolvidos pelos pesquisadores brasileiros.

Em decorrência disso, boa parte do tempo e do trabalho gastos no desenvolvimento de algoritmos que, em geral, são bastante eficientes, é desperdiçada por falta de integração entre aqueles que desenvolvem os programas e os usuários. Para eliminar essa lacuna, neste projeto foi criado um sistema brasileiro voltado para a otimização de problemas através da internet.

Este sistema traz inúmeras vantagens tanto para os usuários dos programas de otimização, como para seus desenvolvedores.

Aos usuários, permite-se escolher, dentre vários programas, aquele que melhor se adequa às características do problema a ser resolvido. Também é possível determinar facilmente se um programa é ou não simples de usar e confiável antes de sua adoção. O usuário tem ainda a comodidade de fazer todos estes testes sem a necessidade de carregar, instalar e ler os manuais dos programas que almeja utilizar. Por fim, muitos programas de otimização exigem, além das funções necessárias para definir o problema, as derivadas

dessas funções. Nesse sentido, neste sistema o usuário pode optar pelo cálculo automático dessas derivadas, geradas pelo pacote Adol-c (vide [9]).

Já para os pesquisadores das universidades, o sistema traz um aumento da visibilidade dos seus programas, além de contribuir para que estes se tornem mais robustos, em função da experiência que será adquirida com sua aplicação a problemas variados.

Esse novo sistema criado, nomeado **Ótimo**, deve servir como ponto de partida para um sistema brasileiro de otimização. Atualmente, este sistema contém dois programas de otimização: O GENCAN, especializado em problemas de programação não-linear de grande porte com restrições de caixa e o ALGENCAN, para problemas de programação não linear de grande porte com restrições gerais. O sistema foi desenvolvido de maneira a tornar relativamente fácil a inclusão de novos programas. O objetivo, a médio prazo, é disponibilizar novos programas no sistema.

Um bom sistema dessa natureza, chamado *NEOS server* (vide [5], [6] e [7]) está disponível em `http://www-neos.mcs.anl.gov/neos`. Esse sistema, bastante completo e eficiente, serviu como inspiração para o Ótimo.

Em seguida, são descritos os próximos capítulos deste trabalho.

### Capítulo 2

Neste capítulo, é feita uma descrição resumida dos componentes básicos, para que o leitor tenha um visão geral do sistema. Também comenta-se sobre os requisitos de *hardware* e *software* para o servidor e o usuário, a fim de garantir o bom funcionamento do sistema.

### Capítulo 3

Um capítulo no qual são descritas as páginas do sistema, incluindo as funções que devem ser enviadas e os parâmetros a serem configurados para cada programa de otimização, assim como os valores que serão considerados padrão.

### Capítulo 4

Aqui, as componentes do sistema são descritas com mais detalhes. Inclui ainda, um pequeno guia de utilização do Adol-C, com um exemplo de como usar essa ferramenta.

**Capítulo 5**

Capítulo em que é feita a simulação do envio de um problema ao Ótimo, incluindo o código dos arquivos enviados e as páginas que são exibidas ao usuário durante o processamento de seu trabalho.

**Capítulo 6**

Este capítulo descreve os arquivos que deverão ser criados e alterados para a inclusão de um novo programa de otimização no Ótimo.

**Capítulo 7**

Capítulo contendo a conclusão e os próximos passos.

**Apêndices**

O Apêndice A contém os arquivos e diretórios do sistema. Já o Apêndice B, exibe as funções do Adol-C usadas pelo Ótimo.

# Capítulo 1

# Descrição geral do sistema

O sistema **Ótimo** serve essencialmente como uma interface via internet entre os programas de otimização e os usuários, sendo responsável pela comunicação entre ambos. Para tal, exige-se que os usuários do sistema possuam os seguintes pré-requisitos:

- Um navegador de internet com o plug-in JavaScript instalado. É através do navegador que o usuário entra no sistema, escolhe um programa e exerce suas preferências. O sistema é compatível com os navegadores Mozilla Firefox, Netscape Navigator e Windows Internet Explorer. O plug-in JavaScript é necessário, já que, no momento da submissão de trabalhos, as escolhas feitas pelos usuários mudam as opções dos programas dinamicamente, o que, no sistema, é feito através de funções escritas em JavaScript. Note que os navegadores atuais já vêm com esse plug-in instalado.

- Um endereço eletrônico (opcional). Com ele, o usuário tem a opção de receber uma notificação informando o término de seu trabalho.

Como o usuário enviará apenas arquivos de tipo texto ao sistema (com a informação das funções que deseja otimizar) e, do mesmo modo, o sistema exibirá apenas arquivos texto ao usuário (contendo a solução), a velocidade da conexão será pouco relevante na submissão de trabalhos. Assim, mesmo usuários com uma conexão lenta não terão problemas para submeter seus trabalhos.

O sistema Ótimo está hospedado em um servidor que tem o UNIX/Linux como sistema operacional e opera com o auxílio dos seguintes *softwares*:

- Servidor de *web* Apache Web Server. Para saber mais sobre o Apache, veja [11].

- Compilador de perl. Todo o sistema está escrito na linguagem Perl, devido as facilidades dessa linguagem para lidar com cadeia de caracteres e páginas HTML. Um boa introdução a essa linguagem pode ser encontrada em [10].

- f2c, um programa que converte funções escritas em Fortran para funções em C/C++. Esse programa é necessário para que o sistema consiga calcular as derivadas automáticas de funções escritas em Fortran, já que o pacote de diferenciação automática deriva apenas funções escritas em C++.

- sendmail, usado para enviar mensagens eletrônicas aos usuários.

- Compiladores de Fortran (g77 e gFortran), C (gcc) e C++ (g++), que são usados pelos programas de otimização que se encontram no sistema.

Para enviar um trabalho, o usuário deve acessar a página do sistema, escolher o programa de otimização, definir os parâmetros e então submeter seu problema. Após a submissão, esse trabalho passará por uma série de etapas controladas pelo sistema até a geração da página com os resultados. A Figura 1.1 exibe o fluxo de dados no sistema.

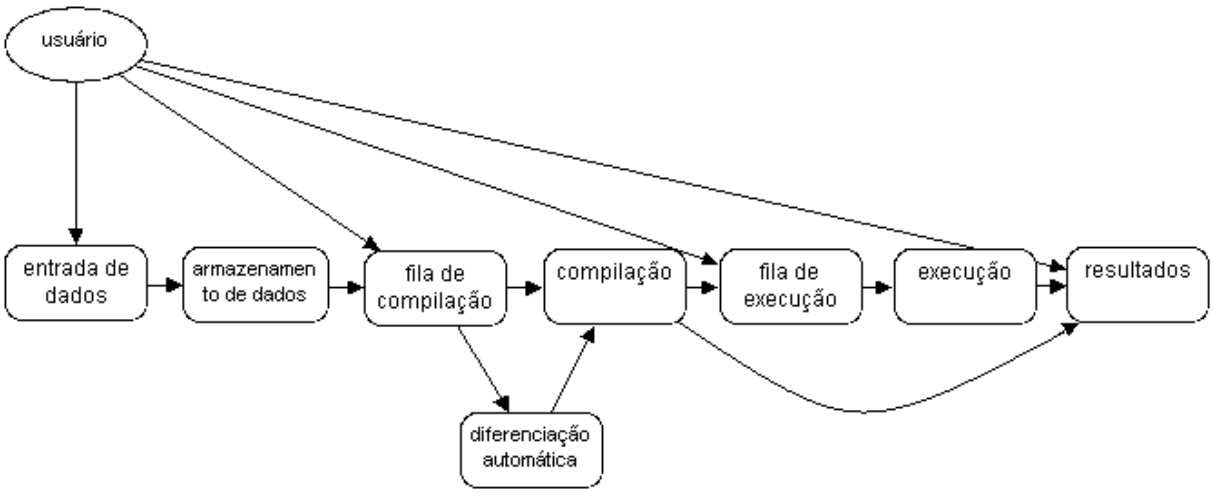



Figura 1.1: Fluxo de dados no sistema

O usuário tem acesso à fase de entrada de dados (envia seu trabalho), às filas (visualização das filas e cancelamento do trabalho) e finalmente, à fase de visualização dos resultados.

O sistema intervêm no processo em todas as etapas que se encontram entre o armazenamento de dados e a apresentação da solução do problema. O trabalho passa pela etapa de diferenciação automática somente se o usuário tiver acionado essa opção no momento da submissão. Por fim, o trabalho pode ir diretamente da etapa de compilação para à de resultados caso o problema enviado apresente erro de compilação.

Nos próximos parágrafos, será feita uma breve descrição dessas etapas. Uma descrição detalhada é apresentada nos capítulos 3 e 4.

- Entrada de dados
  - A página inicial é responsável pelo primeiro contato do usuário com o Ótimo. Ela contém uma lista com os programas de otimização disponíveis e informações a respeito do sistema. Através dela, o usuário pode entrar em contato com os administradores, visualizar o resultado de seu trabalho, cancelar seu trabalho ou ainda acompanhar o andamento das filas.
  - As páginas dos programas de otimização contêm informações específicas sobre o programa e um formulário para que os usuários possam enviar seus arquivos e escolher seus parâmetros.
- Armazenamento de dados
  - Para cada trabalho submetido, é gerado um nome e uma senha. Em seguida, uma pasta é criada no sistema com o nome do trabalho. Nessa pasta, são armazenados todos os arquivos e informações fornecidas pelo usuário.
- Filas
  - Uma fila de trabalhos é necessária a fim de evitar que o funcionamento do sistema seja interrompido devido ao excesso de submissões simultâneas. No entanto, o emprego de uma única fila é inadequado, pois um usuário poderia esperar por horas para ser avisado pelo sistema que seu problema tinha um erro de compilação. Para evitar essa situação, dado que geralmente o tempo de compilação é inferior ao de execução, foram criadas duas filas no sistema. Uma exclusiva para compilação, e outra exclusiva para execução.
- Diferenciação automática
  - O sistema usa o pacote Adol-C para o cálculo de derivadas. Esse pacote calcula derivadas de funções escritas em C/C++. O sistema também pode calcular derivadas de funções em Fortran. Para tal, antes de executar o Adol-C, usa o programa f2c, que converte uma função em Fortran para C.
- Compilação
  - O sistema identifica as linguagens em que foram escritas as funções enviadas, compila estas funções e, em seguida, gera o arquivo executável ao ligá-las com o programa de otimização escolhido.

- Execução
  - O sistema roda o arquivo gerado na fase de compilação.
- Visualização de resultados
  - Pelo Navegador: Quando um trabalho é finalizado, uma página HTML é gerada com a solução encontrada. Caso a solução não tenha sido encontrada, a página conterá a mensagem de erro gerada pelo compilador ou pelo programa de otimização. De posse do nome e senha do trabalho (informações que são enviadas ao usuário na tela, no ato da submissão), o usuário pode acessar a página de resultados. Se não houver filas e o problema do usuário for resolvido rapidamente, a página contendo a solução será aberta instantaneamente no navegador do usuário após a submissão.
  - Por correio eletrônico: O usuário que preencher o campo "email", na página do programa de otimização, receberá duas mensagens do sistema. A primeira, recebida no ato da submissão, informa o nome e a senha do usuário e contém *links* para visualização das filas e cancelamento de trabalhos. A segunda mensagem, a ser recebida após o término do trabalho, informa ao usuário que seu trabalho foi finalizado e fornece um atalho para a página contendo a solução.

# Capítulo 2

# Entrada de dados

Neste capítulo, são exibidas e comentadas as páginas *web* que são responsáveis pela comunicação do sistema com o usuário. São descritas, inclusive, as páginas dos programas de otimização, que contêm informações de como submeter um problema.

Para auxiliar o usuário na submissão de seus trabalhos, foram criadas funções em JavaScript que tornam as páginas dinâmicas, modificáveis pelo usuário, o que também garante a validação dos arquivos enviados. Tais funções encontram-se no arquivo "funcoes.js" e são comentadas ao longo deste capítulo.

Na página inicial, o usuário encontra uma breve apresentação do Ótimo e o seguinte esquema de *links* para acessar as outras páginas do sistema:

- Programas de otimização
  - GENCAN: Problemas de programação não linear de grande porte com restrições de caixa.
  - ALGENCAN 1.0: Problemas de programação não linear gerais de grande porte.
  - ALGENCAN 2.2: Problemas de programação não linear gerais de grande porte.
- Trabalhos
  - Ver filas
  - Ver resultado de trabalho
  - Cancelar trabalho
- Atendimento ao usuário
  - Escreva para o Ótimo

Nas próximas seções, apresenta-se a descrição de cada uma dessas páginas.

## 2.1 GENCAN

O GENCAN é um programa, escrito em Fortran, para a resolução de problemas de programação não linear com restrições de caixa, supondo a diferenciabilidade das funções.

Assim, ele resolve problemas do tipo:

$$\begin{aligned} \min \quad & f(x) \\ \text{suj. a} \quad & l_i \leq x_i \leq u_i, \qquad i = 1, .., n, \end{aligned}$$

tal que $x_i,\ l_i,\ u_i \in \mathbb{R}, \quad f : \mathbb{R}^n \to \mathbb{R}$

O programa foi proposto e implantado por Birgin e Martínez [3], em 2002. Trata-se de um algoritmo que usa o método das restrições ativas, cujo princípio é manter a busca pela solução em uma mesma face, enquanto essa face for promissora. A busca no interior da face é realizada considerando o problema irrestrito cujas variáveis são as variáveis livres dessa face. Assim, as iterações do algoritmo usado para resolver o problema irrestrito, que pode ser do tipo Newton Truncado ou Quase Newton, devem conduzir a um ponto no qual o gradiente interno se anula ou devem parar de maneira abrupta na fronteira da face. Quando o teste padrão indica que a face deve ser abandonada, GENCAN emprega uma iteração do método de Gradiente Projetado Espectral (SPG).

O ALGENCAN, que será explicado na próxima seção, usa o GENCAN para resolver seus subproblemas. Assim, atualmente, o GENCAN é disponibilizado pelo seus desenvolvedores dentro do pacote do ALGENCAN, não podendo ser carregado separamente. Entretanto, dado que o GENCAN é mais simples de usar que o ALGENCAN, para fins didáticos (tanto para usuários como para novos administradores), introduziu-se os pacotes separadamente no sistema Ótimo.

No Ótimo, para resolver seu problema com o GENCAN, o usuário deve enviar ao sistema:

1. Uma rotina que forneça o número de variáveis do problema, um ponto inicial e os limitantes inferiores e superiores. Ou simplesmente fornecer a dimensão, caso prefira utilizar valores padrão para os demais dados.

2. Uma rotina que avalie a função objetivo.

3. Opcionalmente, uma rotina que avalie o gradiente da função objetivo.

4. Opcionalmente, uma rotina que avalie o produto da Hessiana da função objetivo por um vetor. Ou ainda, uma rotina que avalie simplesmente a Hessiana da função objetivo.

5. Finalmente, os valores dos parâmetros do programa. Caso prefira não fornecê-los, valores padrão são usados.

As rotinas fornecidas devem estar escritas em Fortran, C ou C++, sendo possível enviar arquivos de diferentes linguagens em uma mesma submissão. Abaixo, segue a descrição de cada rotina.

**1. Rotina relacionada a dimensão, ponto inicial e limitantes:**

Cabeçalho em Fortran:

```
subroutine inip(n,x,l,u)
integer n
double precision x(n),l(n),u(n)
```

Cabeçalho em C / C++:

```
void inip(int* n,double* x,double* l,double* u)
```

Parâmetros de saída:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - vetor ponto inicial (opcional).
l(n) : real com precisão dupla - vetor de limitantes inferiores (opcional).
u(n) : real com precisão dupla - vetor de limitantes superiores (opcional).

De posse das informações exibidas acima, o usuário já pode enviar a primeira função ao sistema. Para tanto, há um campo na página no qual deve-se fornecer a localização do arquivo a ser enviado. O usuário também tem a opção de ignorar o envio deste arquivo, fornecendo em um outro campo, a dimensão "n" do problema. Neste caso, aos demais parâmetros são atribuídos os seguintes valores:

$$l_i = menos\_inf \quad i = 1..n$$

$$u_i = mais\_inf \quad i = 1..n$$

$$x_i = 0 \qquad i = 1..n$$

de forma que "menos_inf" e "mais_inf" são, respectivamente, o menor e o maior número real armazenado pelo compilador.

O usuário tem ainda a opção de enviar a função "inip" sem fornecer alguns parâmetros (exceto "n"). Neste caso, os valores padrão listados acima são usados. Mesmo assim, todos os parâmetros devem ser declarados.

Quando o usuário preenche o campo "inip", automaticamente o sistema desabilita o campo "n". A função "JS3" em "funcoes.js" é responsável por essa validação. Do mesmo modo, o sistema desabilita o campo "inip" quando o campo "n" é selecionado (função "JS4"). Essas funções, escritas em JavaScript, deixam mais claro para o usuário quais arquivos devem ser enviados, além de diminuir a possibilidade de erro por envio de arquivos conflitantes.

Finalmente, se o usuário tentar submeter sem preencher ambos os campos "inip" e "n", um alerta é exibido indicando a falta do arquivo e impedindo a submissão (função "JS2").

**2. Rotina que calcula a função objetivo:**

Cabeçalho em Fortran:

```
subroutine evalf(n,x,f,flag)
integer n,flag
double precision x(n),f
```

Cabeçalho em C / C++:

```
void evalf(int* n,double* x, double* f,int* flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - ponto corrente.

Parâmetros de saída:

f : real com precisão dupla - valor da função objetivo em x.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

Respeitando as declarações acima, basta o usuário fornecer a localização do arquivo no campo pedido para completar o envio da função. Obviamente, o envio desta rotina é obrigatório e, caso o usuário tente submeter seu problema sem o envio da mesma, uma

mensagem de alerta é exibida e a submissão cancelada (pela função "JS2" do sistema).

**3. Rotina que calcula o gradiente da função objetivo:**

O usuário pode escolher entre enviar o arquivo com a função, usar aproximação por diferenças finitas (fornecidas pelo próprio GENCAN), ou ainda, usar um pacote de diferenciação automática (veja a Seção 3.5). O conteúdo da página muda a cada vez que o usuário troca de opção, a fim de exibir informações referentes à opção selecionada. Essa mudança é feita pela função "JS6". Caso o usuário opte por enviar a função, a mesma deve seguir o formato especificado abaixo:

Cabeçalho em Fortran:

```
subroutine evalg(n,x,g,flag)
integer n,flag
double precision x(n),g(n)
```

Cabeçalho em C / C++:

```
void evalg(int* n,double* x, double* g,int* flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - ponto corrente.

Parâmetros de saída:

g(n) : real com precisão dupla - gradiente da função objetivo em x.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

**4. Rotinas que envolvem as derivadas de segunda ordem da função objetivo:**

O usuário pode escolher entre enviar o arquivo com a função que calcula o produto da Hessiana da função objetivo por um vetor, ou um arquivo que calcula somente a Hessiana. Pode, ainda, usar o pacote de diferenciação automática para o cálculo do produto Hessiana-vetor ou para o cálculo somente da Hessiana. Finalmente, pode usar aproximação por diferenças finitas. Novamente, o conteúdo da página muda cada vez que o usuário troca de opção (segundo a função "JS7").

Caso o usuário opte por enviar a função que calcula o produto da Hessiana da função objetivo por um vetor, a mesma deve seguir o formato especificado abaixo:

**Produto da Hessiana da função objetivo por um vetor:**

Cabeçalho em Fortran:

```
subroutine evalhlp(n,x,p,hp,goth,flag)
integer n,flag
double precision x(n),p(n),hp(n)
logical goth
```

Cabeçalho em C / C++:

```
void evalhlp(int n,double *x,double *p,double *hp,int *goth,int *flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - ponto corrente.
p(n) : real com precisão dupla - vetor a ser multiplicado pela Hessiana.
goth : lógico (.true. ou .false.) em Fortran e inteiro (1 ou 0) em C - parâmetro usado para indicar se houve alteração no ponto corrente. O algoritmo de otimização atribui o valor "false" toda vez que o ponto corrente é modificado. Se o valor de entrada for "false", o usuário pode salvar a Hessiana em um bloco de variáveis globais ("common structure", no Fortran) e fazer "goth" receber "true". Sendo assim, é possível usar a chamada novamente sem que haja necessidade de recalcular a Hessiana.

Parâmetros de saída:

hp(n) : real com precisão dupla - produto da Hessiana pelo vetor.
goth : lógico (.true. ou .false.) em Fortran e inteiro (1 ou 0) em C - ver definição acima.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a execução da função e zero em caso contrário.

Caso o usuário opte por enviar a função que calcula a Hessiana da função objetivo, a mesma deve seguir o formato especificado abaixo:

**Hessiana da função objetivo:**

Cabeçalho em Fortran:

```
subroutine evalh(n,x,hlin,hcol,hval,nnzh,flag)
integer n,hlin(*),hcol(*),nnzh,flag
double precision hval(*),x(n)
```

Cabeçalho em C / C++:

```
void evalh(int n,double* x,int* hlin,int* hcol,double* hval,int* nnzh,
int* flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis
x(n) : real com precisão dupla - ponto corrente

Parâmetros de saída:

nnzh : inteiro - número de elementos não nulos da Hessiana (somente a parte triangular inferior).
hlin(nnzh) : inteiro - índices das linhas dos elementos não nulos da Hessiana.
hcol(nnzh) : inteiro - índices das colunas dos elementos não nulos da Hessiana.
hval(nnzh) : real com precisão dupla - valores dos elementos não nulos da Hessiana.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

Essa representação dos elementos não nulos da matriz por coordenadas é comum nos programas de otimização, sendo útil principalmente para tornar mais eficiente as operações computacionais com matrizes esparsas. Como exemplo, suponha que a matriz abaixo deva ser representada no formato mencionado.

$$\mathbf{H} = \begin{pmatrix} 7 & 5 \\ 5 & 0 \end{pmatrix}$$

Como as matrizes Hessianas enviadas serão simétricas (dado que as segundas derivadas da função objetivo são contínuas, por premissa do algoritmo de otimização), é suficiente armazenar a parte triangular inferior. Assim, neste exemplo, apenas os elementos 7 (linha 1, coluna 1) e 5 (linha 2, coluna 1) precisam ser armazenados. Em Fortran, o código seria:

```
nnzh = 2
hlin(1) = 1
hcol(1) = 1
hval(1) = 7
hlin(2) = 2
hcol(2) = 1
hval(2) = 5
```

Deve-se lembrar que, caso o código fosse escrito em C, o vetor começaria no índice "0".

**5. Parâmetros do programa:**

Os parâmetros são configurados diretamente na página, através de campos de texto do formulário de submissão. Abaixo, estão os parâmetros do GENCAN com seus respectivos valores padrão. Entre parênteses, consta o nome do campo na versão sem detalhes da página do GENCAN (veja pág 17):

- Usar precondicionador para os gradientes conjugados (o precondicionador é uma correção quase-Newton da aproximação Gauss-Newton da Hessiana, conforme [4]) (precond): Sim.
- Tolerância de otimalidade para a norma infinito do gradiente projetado (epsopt): 0.00001.
- Número máximo de iterações (maxtotit): 10000.
- Número máximo de avaliações da função objetivo (maxtotfc): 100000.
- Número de componentes dos vetores exibidos na solução detalhada (veja próximo parágrafo) (ncomp): 5.

Além das opções acima, através de um botão na página, é possível gerar uma saída detalhada com informações sobre as iterações. O usuário pode escolher entre uma saída apenas com informações básicas sobre as iterações ou ainda uma saída que forneça, além das informações básicas, informações das buscas lineares e do cálculo da direção em cada iteração interna. A função "JS10" é a responsável por exibir as informações para cada tipo de saída escolhida.

Caso o usuário deseje salvar os parâmetros listados acima, para poder usá-los em trabalho posterior, basta acionar a opção "salvar parâmetros". Atente que, para posterior visualização dos parâmetros salvos, o usuário deve habilitar a opção "aceitar *cookies*" do seu navegador. A função "JS13" é a responsável por recuperar os dados salvos em uma submissão, ou seja, é responsável por ler os *cookies*. No entanto, os *cookies* não são gerados por uma função JavaScript, e sim por meio dos arquivos CGI responsáveis pelo armazenamento de dados, o que será visto mais adiante.

Por fim, o último campo que pode ser preenchido é o "receber os resultados via email". Ao preenchê-lo, o usuário receberá duas mensagens ao longo do processamento de seu trabalho.

No topo da página do GENCAN, também há um *link* denominado "ver página sem detalhes" que leva a uma outra página de submissão do GENCAN, que não contém informações sobre os arquivos a serem enviados. Assim, um usuário experiente pode ter acesso

a uma página mais "limpa", possibilitando que submeta mais rapidamente seus problemas. Essa página é controlada pelo arquivo "funcoes_det.js", cujas funções são semelhantes às já descritas em "funcoes.js". Na Figura 2.1, segue a página sem detalhes para o GENCAN.

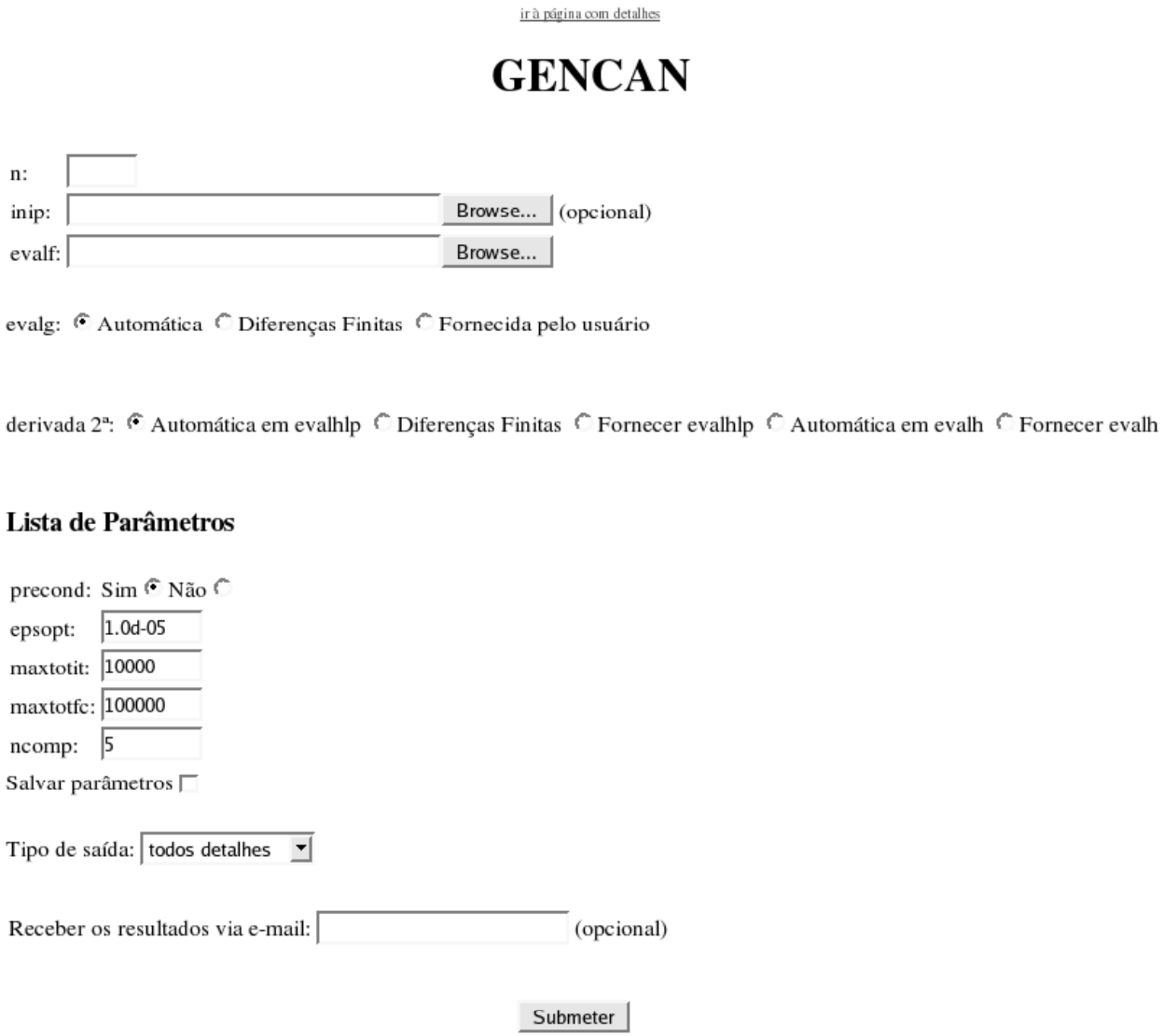



Figura 2.1: Página sem detalhes do GENCAN.

## 2.2 ALGENCAN

O ALGENCAN é um programa, escrito em Fortran, para resolução de problemas gerais de programação não linear.

Assim, o ALGENCAN resolve problemas do tipo:

$$\begin{aligned} \min \quad & f(x) \\ \text{suj} \quad & c_i(x) = 0 \qquad i \in I, \\ & g_j(x) \leq 0 \qquad j \in D, \\ & l_i \leq x_i \leq u_i, \qquad i = 1, .., n \end{aligned}$$

tal que

$x_i,\ l_i,\ u_i \in \mathbb{R}, \quad f,\ c_i,\ g_j \colon \mathbb{R}^n \to \mathbb{R}.$

$I$: Conjunto das restrições de igualdade.

$D$: Conjunto das restrições de desigualdade.

O método utilizado pelo programa é do tipo Lagrangiano Aumentado, cujo princípio consiste em gerar uma sequência de minimizações (aproximadas) da função Lagrangiano Aumentado, sujeitas somente às restrições "simples" do problema. As restrições que não são "simples" são introduzidas na função Lagrangiano através de termos de penalização, de modo que a minimização do Lagrangiano com as restrições "simples" possa ser realizada por um algoritmo especializado. No ALGENCAN, as restrições "simples" são as restrições de caixa e o algoritmo especializado utilizado para resolver tais subproblemas é o GENCAN (mencionado na seção anterior). Se, entre duas minimizações do Lagrangiano (iterações externas), não houver progresso em termos de admissibilidade ou complementaridade, o parâmetro de penalização é aumentado. Caso contrário, o valor do parâmetro permanece o mesmo. Para mais detalhes, veja [1] e [2].

No Ótimo, para resolver seu problema com o ALGENCAN[1], o usuário deve enviar ao sistema:

1. Uma rotina que forneça a dimensão do problema, o ponto inicial, os limitantes inferiores e superiores, o número de restrições, valores iniciais para os multiplicadores de Lagrange, o tipo de cada restrição (igualdade ou desigualdade) e informações sobre a linearidade das restrições. Opcionalmente, é possível fornecer simplesmente a dimensão e o número de restrições, caso se prefira utilizar os valores padrão.

[1] Dados referentes à versão 2.2 do ALGENCAN.

2. Uma rotina que avalie a função objetivo e uma rotina que avalie as restrições. Ou ainda, uma única rotina que avalie a função objetivo e as restrições.

3. Opcionalmente, uma rotina que avalie o gradiente da função objetivo e uma rotina que avalie os gradientes das restrições. Ou ainda, uma única rotina que avalie o gradiente da função objetivo e os gradientes das restrições.

4. Opcionalmente, uma rotina que avalie a Hessiana do Lagrangiano multiplicada por um vetor, ou uma rotina que avalie a Hessiana do Lagrangiano, ou ainda, rotinas separadas que avaliem a Hessiana da função objetivo e a Hessiana das restrições.

5. Finalmente, os valores dos parâmetros do programa. Caso o usuário prefira não fornecer esse parâmetros, valores padrão serão usados.

As rotinas fornecidas devem estar escritas em Fortran, C ou C++. Diferentemente do GENCAN, todas rotinas devem estar escritas na mesma linguagem. Abaixo, segue a descrição de cada rotina.

**1. Rotina relacionada a dimensão, valores iniciais, limitantes e restrições:**

Cabeçalho em Fortran:

```
subroutine inip(n,x,l,u,m,lambda,equatn,linear)
integer n,m
double precision x(n),l(n),u(n),lambda(m)
logical equatn(m),linear(m)
```

Cabeçalho em C / C++:

```
void inip(int* n,double** x,double** l,double** u,int* m,double** lambda,
int** equatn,int** linear)
```

Parâmetros de saída:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - ponto inicial (opcional).
l(n) : real com precisão dupla - vetor de limitantes inferiores (opcional).
u(n) : real com precisão dupla - vetor de limitantes superiores (opcional).
m : inteiro - número de restrições (sem incluir as canalizações).
lambda(m) : real com precisão dupla - estimativa inicial dos multiplicadores de Lagrange (opcional).
equatn(m) : lógico (.true. ou .false.) em Fortran e inteiro (1 ou 0) em C - indica se cada restrição é de igualdade (.true. ou 1) ou de desigualdade (.false. ou 0) (opcional).
linear(m) : lógico (.true. ou .false.) em Fortran e inteiro (1 ou 0) em C - indica se cada

restrição é linear (.true. ou 1) ou não linear (.false. ou 0) (opcional).

De posse das informações exibidas acima, o usuário já pode enviar a primeira função ao sistema. Para tal, há um campo na página no qual se deve fornecer a localização do arquivo a ser enviado. O usuário também tem a opção de ignorar o envio deste arquivo, preenchendo dois outros campos. O primeiro refere-se a dimensão e o segundo ao número de restrições do problema. Neste caso, os parâmetros são iniciados com os seguintes valores:

$$
\begin{aligned}
l_i &= menos\_inf \qquad i = 1, .., n \\
u_i &= mais\_inf \qquad i = 1, .., n \\
x_i &= 0 \qquad i = 1..n \\
lambda_j &= 0 \qquad j = 1, .., m \\
equatn_j &= false \qquad j = 1, .., m \\
linear_j &= false \qquad j = 1, .., m
\end{aligned}
$$

de forma que "menos_inf" e "mais_inf" são, respectivamente, o menor e o maior número real armazenado pelo compilador.

O usuário tem ainda a opção de enviar a função "inip" sem iniciar alguns parâmetros (exceto "n" e "m"). Para os parâmetros não enviados, os valores padrão listados acima são usados. Mesmo assim, todos os parâmetros devem ser declarados.

De forma análoga ao GENCAN, no ALGENCAN, quando o usuário preenche o campo "inip", o sistema desabilita automaticamente os campos "n" e "m". A função "JS3" em "funcoes.js" é reponsável por essa validação. Do mesmo modo, o sistema desabilita o campo "inip" quando os campos "n" e "m" são selecionados (função "JS5"). Essas funções em JavaScript deixam mais claro para o usuário quais arquivos devem ser enviados, além de diminuir a possibilidade de erro por envio de arquivos conflitantes.

Finalmente, se o usuário tenta submeter seu problema sem preencher os campos "inip", "n" e "m", um alerta é exibido indicando a falta do arquivo e impedindo a submissão (função "JS1").

**2. Rotinas que calculam a função objetivo e as restrições:**

O usuário pode escolher entre enviar rotinas separadas para o cálculo da função objetivo e das restrições, ou enviar uma única rotina. A função JS16 faz o controle dessa

escolha, enquanto a função JS1 manda um alerta caso o usuário tenha submetido seu problema sem ter enviado nenhuma delas.

Caso o usuário opte por enviar separadamente as duas rotinas, deve seguir o formato especificado abaixo:

**Função objetivo**

Cabeçalho em Fortran:

```
subroutine evalf(n,x,f,flag)
integer n,flag
double precision x(n),f
```

Cabeçalho em C / C++:

```
void evalf(int* n,double* x, double* f,int* flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - ponto corrente.

Parâmetros de saída:

f : real com precisão dupla - valor da função objetivo em x.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

**Restrições**

Cabeçalho em Fortran:

```
subroutine evalc(n,x,ind,c,flag)
integer n,ind,flag
double precision x(n),c
```

Cabeçalho em C / C++:

```
void evalc(int n,double* x,int ind,double* c,int* flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.

x(n) : real com precisão dupla - ponto corrente.
ind : inteiro - índice da restrição a ser calculada.

Parâmetros de saída:

c : real com precisão dupla - valor da ind-ésima restrição em x.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

Note que "evalc" calcula apenas uma restrição, cujo o índice "ind" é passado na entrada. Por exemplo, suponha que se deseje codificar "evalc" para as seguintes restrições:

$$c_1(x) = x_1^2 + x_2$$
$$c_2(x) = 2x_2$$

O trecho do código, em Fortran, seria:

```
if ( ind .eq. 1 ) then
  c = x(1)**2 + x(2)
else if ( ind .eq. 2 ) then
  c = 2*x(2)
end if
return
```

Caso o usuário opte por enviar uma única rotina, deve seguir o formato especificado abaixo:

**Função objetivo e restrições na mesma rotina**

Cabeçalho em Fortran:

```
subroutine evalfc(n,x,f,m,c,flag)
integer flag,m,n
double precision f,c(m),x(n)
```

Cabeçalho em C / C++:

```
void evalfc(int n,double *x,double *f,int m,double *c,int *flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.

m : inteiro - número de restrições.
x(n) : real com precisão dupla - ponto corrente.

Parâmetros de saída:

f : real com precisão dupla - valor da função objetivo em x.
c(m) : real com precisão dupla - valor da m-ésima restrição em x.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

Note que "evalfc" calcula todas restrição de uma vez, e as guarda no vetor "c(m)".

**3. Rotinas que calculam o gradiente da função objetivo e o gradiente das restrições:**

O usuário poderá escolher entre enviar as duas rotinas separadamente, enviar uma única rotina, usar aproximação por diferenças finitas, ou usar um pacote de diferenciação automática. A escolha do usuário entre essas quatro alternativas afeta as opções do próximo item, referente às derivadas de segunda ordem. A função JS17 faz o controle dessas escolhas.

Caso o usuário opte por enviar as duas rotinas separadamente, estas devem seguir o formato especificado abaixo:

**Gradiente da função objetivo**

Cabeçalho em Fortran:

```
subroutine evalg(n,x,g,flag)
integer n,flag
double precision x(n),g(n)
```

Cabeçalho em C / C++:

```
void evalg(int* n,double* x, double* g,int* flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - ponto corrente.

Parâmetros de saída:

g(n) : real com precisão dupla - valor do gradiente da função objetivo em x.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

**Gradiente das restrições**

Cabeçalho em Fortran:

```
subroutine evaljac(n,x,ind,jcvar,jcval,jcnnz,flag)
integer n,ind,jcvar(n),jcnnz,flag
double precision x(n),jcval(n)
```

Cabeçalho em C / C++:

```
void evaljac(int n,double* x,int ind,int* jcvar,double* jcval,
int* jcnnz,int* flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - ponto corrente.
ind : inteiro - índice da restrição cujo gradiente será calculado.

Parâmetros de saída:

jcnnz : inteiro - número de possíveis elementos não nulos do gradiente.
jcvar(jcnnz) : inteiro - coordenadas dos elementos não nulos do gradiente.
jcval(jcnnz) : real com precisão dupla - valor do gradiente usando como coordenadas os índices do vetor jcvar.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

Analogamente a "evalc", "evaljac" calcula o gradiente de uma única restrição, cujo índice "ind" é passado na entrada. Além disso, essa função usa a representação de vetor esparso para o armazenamento dos gradientes, a fim de tornar mais eficiente as manipulações algébricas.

Por exemplo, suponha que se deseje codificar "evaljac" para as mesmas restrições apresentadas na pág. 22. Neste caso, o trecho do código, em Fortran, seria:

```
if ( ind .eq. 1 ) then
  jcnnz = 2
  jcvar(1) = 1
```

```
  jcval(1) = 2 * x(1)
  jcvar(2) = 2
  jcval(2) = 1
else if ( ind .eq. 2 ) then
  jcnnz = 1
  jcvar(1) = 2
  jcval(1) = 2
end if
```

Caso o usuário opte por enviar uma única rotina, deve seguir o formato especificado abaixo:

**Gradiente da função objetivo e das restrições na mesma rotina**

Cabeçalho em Fortran:

```
subroutine evalgjac(n,x,g,m,jcfun,jcvar,jcval,jcnnz,flag)
integer flag,jcnnz,m,n,i,jcfun(*),jcvar(*)
double precision g(n),jcval(*),x(n)
```

Cabeçalho em C / C++:

```
void evalgjac(int n,double *x,double *g,int m,int *jcfun,
int *jcvar,double *jcval,int *jcnnz,int *flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.
m : inteiro - número de restrições.
x(n) : real com precisão dupla - ponto corrente.

Parâmetros de saída:

g(n) : real com precisão dupla - valor do gradiente da função objetivo em x.
jcnnz : inteiro - número de possíveis elementos não nulos da matriz Jacobiana.
jcfun(jcnnz) : inteiro - índice referente à restrição.
jcvar(jcnnz) : inteiro - índice referente à variável.
jcval(jcnnz) : real com precisão dupla - valor da derivada da restrição jcfun em relação à variável jcvar.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

Por exemplo, suponha que se deseje codificar "evalgjac" para as mesmas restrições apresentadas na pág. 22. Suponha, também, que a função objetivo seja constante. Neste caso, o trecho do código, em Fortran, seria:

```
Parte referente ao gradiente da função objetivo:
g(1)=0
g(2)=0

Parte referente à Jacobiana das restrições:
jcnnz = 3
jcfun(1) = 1
jcvar(1) = 1
jcval(1) = 2 * x(1)
jcfun(2) = 1
jcvar(2) = 2
jcval(2) = 1
jcfun(3) = 2
jcvar(3) = 2
jcval(3) = 2
```

Note que "evalgjac" calcula todos os gradientes das restrições de uma só vez, diferente do que acontece com "evaljac", que calcula um gradiente por vez.

**4. Rotinas para as derivadas de segunda ordem:**

Cada escolha feita para a derivada de primeira ordem possibilita diferentes opções para as derivadas de segunda ordem, como mostrado a seguir.

- Derivada de primeira ordem por diferenciação automática. Neste caso, para a derivada de segunda ordem, pode-se optar por:
  - Usar diferenciação automática. Através do Adol-C, obtém-se o produto da Hessiana do Lagrangiano por um vetor.
  - Aproximar as derivadas de segunda ordem por diferenças finitas.
- Derivada de primeira ordem por diferenças finitas. Neste caso, as derivadas de segunda ordem são aproximadas por diferenças finitas.
- Derivada de primeira ordem fornecidas pelo usuário. Neste caso, pode-se optar por:
  - Fornecer separadamente a matrix Hessiana da função objetivo e a Hessiana das restrições.

- Fornecer o produto da Hessiana do Lagrangiano por um vetor.
- Fornecer a Hessiana do Lagrangiano.
- Usar diferenciação automática para se obter o produto da Hessiana do Lagrangiano por um vetor.
- Aproximar as derivadas de segunda ordem por diferenças finitas.

A função JavaScript que controla o conteúdo exibido na página, dada uma escolha para a derivada de primeira ordem, é "JS17". Já a função "JS19", controla o conteúdo exibido para cada opção escolhida da derivada de segunda ordem. Por fim, se o usuário selecionar diferenças finitas para a derivada de segunda ordem, aparecerá um novo botão com as opções para o tipo de diferenças finitas a ser usado. Tal controle é feita pela função "JS18".

Caso o usuário tenha escolhido enviar as rotinas com a derivada de primeira ordem no item 3 e tenha decidido enviar separadamente a Hessiana da função objetivo e a Hessiana das restrições, deve seguir os formatos abaixo:

**Hessiana da função objetivo:**

Cabeçalho em Fortran:

```
subroutine evalh(n,x,hlin,hcol,hval,hnnz,flag)
integer n,hlin(*),hcol(*),hnnz,flag
double precision hval(*),x(n)
```

Cabeçalho em C / C++:

```
void evalh(int n,double* x,int* hlin,int* hcol,double* hval,int* hnnz,
int* flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - ponto corrente.

Parâmetros de saída:

hnnz : inteiro - número de possíveis elementos não nulos da Hessiana (somente a parte triangular inferior).
hlin(hnnz) : inteiro - índice da linha da Hessiana.
hcol(hnnz) : inteiro - índice da coluna da Hessiana.
hval(hnnz) : real com precisão dupla - valores dos elementos não nulos da Hessiana usando como coordenadas os índices de hlin e hcol.

flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

Na pág 15, há um exemplo de como representar uma matriz usando esse formato.

**Hessiana das restrições**

Cabeçalho em Fortran:

```
subroutine evalhc(n,x,ind,hclin,hccol,hcval,hcnnz,flag)
integer n,ind,hclin(*),hccol(*),hcnnz,flag
double precision hcval(*),x(n)
```

Cabeçalho em C / C++:

```
void evalhc(int n,double* x,int ind,int* hclin,int* hccol,double* hcval,
int* hcnnz,int* flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - ponto corrente.
ind : inteiro - índice da restrição cuja Hessiana será calculada.

Parâmetros de saída:

hcnnz : inteiro - número de possíveis elementos não nulos da Hessiana (somente a parte triangular inferior).
hclin(hcnnz) : inteiro - índice da linha da Hessiana.
hccol(hcnnz) : inteiro - índice da coluna da da Hessiana.
hcval(hcnnz) : real com precisão dupla - valores dos elementos não nulos da Hessiana usando como coordenadas os índices de hclin e hccol.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

De maneira similar ao que ocorre com "evalc" e "evaljac", "evalhc" calcula a Hessiana de uma restrição por vez, dado o índice "ind" passado na entrada. Novamente, usa-se a representação de vetor esparso para o armazenamento dos gradientes, a fim de tornar mais eficiente as manipulações algébricas quando os vetores são esparsos (veja pág. 15).

Caso o usuário tenha escolhido enviar as rotinas com a derivada de primeira ordem no item 3 e tenha escolhido enviar o produto da Hessiana do Lagrangiano por um vetor,

a função deve seguir o formato abaixo:

**Produto da Hessiana do Lagrangiano por um vetor**

Cabeçalho em Fortran:

```
subroutine evalhlp(n,x,m,lambda,sf,sc,p,hp,goth,flag)
integer n,m,flag
double precision x(n),lambda(m),p(n),hp(n),sf,sc(m)
logical goth
```

Cabeçalho em C / C++:

```
void evalhlp(int n,double *x,int m,double *lambda,double sf,
double *sc,double *p,double *hp,int *goth,int *flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - ponto corrente.
m : inteiro - número de restrições (não incluir bordas).
lambda(m) : real com precisão dupla - vetor de multiplicadores de Lagrange.
p(n) : real com precisão dupla - vetor a ser multiplicado pela matriz.
sf: real com precisão dupla - fator de escalamento da função objetivo.
sc(m): real com precisão dupla - fatores de escalamento das restrições.
goth : lógico (.true. ou .false.) em Fortran e inteiro (1 ou 0) em C - parâmetro usado para indicar se houve necessidade de recalcular a Hessiana. O algoritmo de otimização atribui o valor "false" toda vez que o ponto corrente é modificado. Se o valor de entrada for "false", o usuário pode salvar a Hessiana em um bloco de variáveis globais ("common structure", no Fortran) e fazer "goth" receber "true". Sendo assim, é possível usar a chamada novamente sem que haja necessidade de recalcular a Hessiana.

Parâmetros de saída:

hp(n) : real com precisão dupla - produto da Hessiana pelo vetor.
goth : lógico (.true. ou .false.) em Fortran e inteiro (1 ou 0) em C - ver definição acima.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

Caso o usuário tenha escolhido enviar as rotinas com a derivada de primeira ordem no item 3 e tenha escolhido enviar a Hessiana do Lagrangiano, a função deve seguir o formato abaixo:

**Hessiana do Lagrangiano**

Cabeçalho em Fortran:

```
subroutine evalhl(n,x,m,lambda,scalef,scalec,hllin,hlcol,hlval,hlnnz,flag)
integer flag,hlnnz,m,n,hlcol(*),hllin(*)
double precision scalef,hlval(*),lambda(m),scalec(m),x(n)
```

Cabeçalho em C / C++:

```
void evalhl(int n,double *x,int m,double *lambda,double scalef,
double *scalec,int *hllin,int *hlcol,double *hlval,int *hlnnz,
int *flag)
```

Parâmetros de entrada:

n : inteiro - número de variáveis.
x(n) : real com precisão dupla - ponto corrente.
m : inteiro - número de restrições (não incluir bordas).
lambda(m) : real com precisão dupla - vetor de multiplicadores de Lagrange.
scalef: real com precisão dupla - fator de escalamento da função objetivo.
scalec(m): real com precisão dupla - fatores de escalamento das restrições.

Parâmetros de saída:

hlnnz : inteiro - número de possíveis elementos não nulos da Hessiana do Lagrangiano (somente a parte triangular inferior).
hllin(hlnnz) : inteiro - índice da linha da Hessiana do Lagrangiano.
hlcol(hlnnz) : inteiro - índice da coluna da Hessiana do Lagrangiano.
hlval(hlnnz) : real com precisão dupla - valores dos elementos não nulos da Hessiana do Lagrangiano, usando como coordenadas os índices de hllin e hlcol.
flag : inteiro - deve ser diferente de zero se ocorrer algum erro durante a avaliação da função e zero em caso contrário.

**6. Parâmetros do ALGENCAN:**

Os parâmetros do ALGENCAN são configurados diretamente na página, através de campos de texto no formulário de submissão. Abaixo, estão os parâmetros com seus respectivos valores padrão. Entre parênteses, consta o nome do campo na versão sem detalhes na página do ALGENCAN:

- Pular passo para aceleração do algoritmo (skipacc): Sim.
- Equilibrar escala das variáveis dos sistemas lineares (sclsys):Não.

- Remover variáveis fixas (ou seja, variáveis com limitantes inferiores e superiores iguais)(rmfixv): Sim.

- Variáveis de folga para as restrições de desigualdade (slacks): Não.

- Equilibrar escala da função objetivo e restrições (scale): Sim.

- Ignorar a função objetivo (para achar apenas uma solução viável)(ignoref): Não.

- Verificar a derivada das funções enviadas (comparação com diferenças finitas)(checkder): Não.

- Tolerância de factibilidade (para a norma infinito das restrições) (epsfeas): 1.0d-04.

- Tolerância de otimalidade (para a norma infinito do gradiente projetado do Lagrangiano Aumentado) (epsopt): 1.0d-04.

- Número de componentes dos vetores exibidos na solução detalhada. Veja próximo parágrafo (ncomp): 5.

Após a escolha dos parâmetros, o usuário deve definir o nível de detalhe com que a solução encontrada pelo programa de otimização será apresentada. Para tal, deve-se definir no campo "Informação externas" o nível de detalhe, em uma escala que vai de zero (sem informações das iterações externas) até cinco (todas informações geradas pelo programa). Analogamente, deve-se definir o nível de detalhe das "Informações internas", em uma escala que vai de zero a seis.

Caso o usuário deseje salvar os parâmetros listados acima, para poder usá-los em trabalho posterior, basta acionar a opção "salvar parâmetros". Atente que, para posterior visualização dos parâmetros salvos, o usuário deve habilitar a opção "aceitar *cookies*" do seu navegador. A função "JS14" é a responsável por recuperar os dados salvos para o ALGENCAN, ou seja, é responsável por ler os *cookies*.

Por fim, o último campo que pode ser preenchido é denominado "receber os resultados via email". Ao preenchê-lo, o usuário receberá duas mensagens ao longo do processamento de seu trabalho.

Analogamente ao GENCAN (veja pág 17), também foi criada uma página menos detalhada para o ALGENCAN, destinada a usuários mais experientes. Essa página é mostrada na Figura 2.2.

# ALGENCAN

n: m:

inip: /home/otimo/entradas/algencan2/inip_novo.f Browse... (opcional)

Evalf e Evalc Evalfc

evalf: /home/otimo/entradas/algencan2/avalia_f.f Browse...

evalc: /home/otimo/entradas/algencan2/avalia_c.f Browse...

Derivada 1ª:

Evalg e Evaljac Evalgjac Diferenças Finitas Automática

Derivada 2ª:

Será calculada por Diferenças Finitas.

## Lista de Parâmetros

skipacc: Sim Não

sclsys: Sim Não

rmfixv: Sim Não

slacks: Sim Não

scale: Sim Não

ignoref: Sim Não

checkder: Sim Não

epsfeas: 0.0001

epsopt: 0.0001

ncomp: 5

Informações externas: 1 Informações internas: 0

Salvar parâmetros

Receber os resultados via e-mail: (opcional)

Submeter

Figura 2.2: Página sem detalhes do ALGENCAN.

## 2.3 Ver filas

Esta página exibe as duas filas do sistema, a de compilação e a de execucão. Em cada fila, constam o trabalho que está sendo realizado naquele instante e os próximos a serem realizados. O trabalho corrente fica no topo da fila e recebe uma indicação "compilando" ou "executando", a depender do tipo da fila. Os trabalhos em espera ficam abaixo do corrente, ordenados pela ordem de chegada. O texto "vazia" em uma fila indica que não há trabalhos sendo processados na mesma. Na Figura 2.3, é mostrado um exemplo da página de visualização das filas.

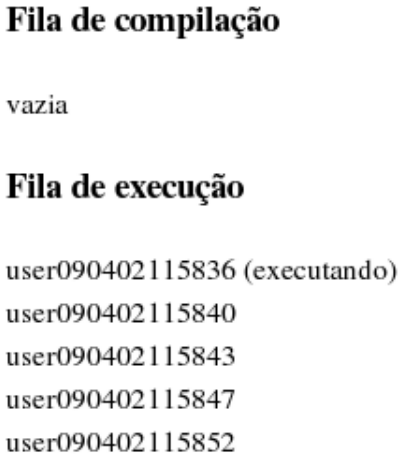



Figura 2.3: Página de visualização das filas

## 2.4 Ver resultado de trabalho

Nesta página, o usuário deve entrar com o nome e a senha de seu trabalho, a fim de visualizar o resultado alcançado pelo programa de otimização. O nome e a senha são passados para o usuário assim que o trabalho é submetido, sendo também enviados para seu endereço eletrônico, caso este tenha sido fornecido. O trabalho fica gravado no sistema por algumas semanas e então é apagado. A Figura 2.4 exibe um exemplo dessa página.

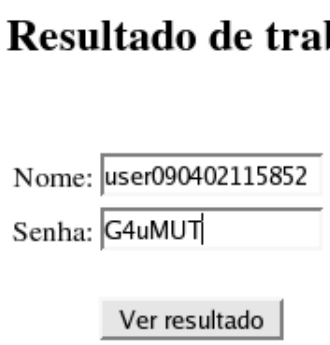



Figura 2.4: Página para acesso à visualização dos resultados

A página com os resultados é exibida no capítulo 5, na Figura 4.4. Nesse capítulo, um exemplo de submissão de problema é mostrado passo à passo.

## 2.5 Cancelar trabalho

De forma análoga à página anterior, para cancelar um trabalho o usuário deve fornecer o nome e a senha do mesmo. Nesse caso, não é gerado um arquivo de saída para o problema. Um trabalho será automaticamente cancelado caso esteja sendo processado pelo sistema por mais de 24 horas. A página para o cancelamento de um problema é idêntica àquela para a visualização dos resultados.

## 2.6 Escreva para o Ótimo

O usuário deve acessar esta página para entrar em contato com o administrador do sistema Ótimo. Há um espaço para o usuário escrever suas dúvidas, comentários e sugestões. Há também os campos "nome" e "email" que podem ser opcionalmente preenchidos pelo usuário. Na Figura 2.5, exibe-se essa página.

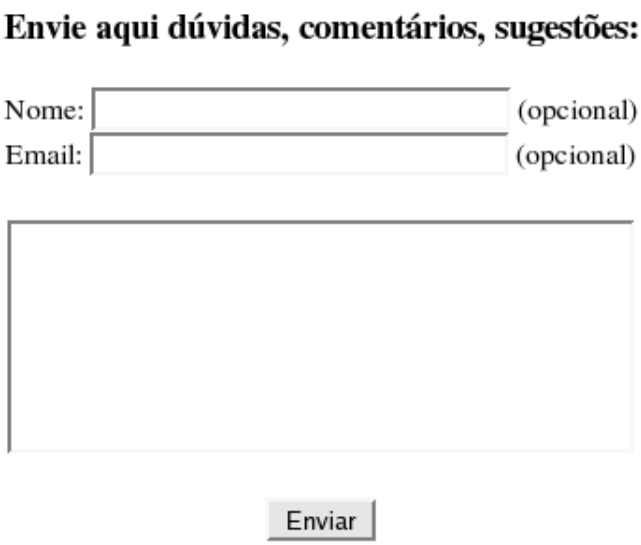



Figura 2.5: Página para o envio de sugestões

# Capítulo 3

# Funcionamento do sistema

No capítulo anterior, foi descrita a entrada de dados, primeira etapa do fluxo de dados no sistema, apresentado na Figura 1.1. As outras etapas desse fluxo são detalhadas nesta seção.

## 3.1 Armazenamento de dados

Quando o usuário clica no botão "submeter", na página de um programa de otimização, são acionadas as primeiras verificações de consistência, via funções em JavaScript, já descritas no capítulo anterior. Após a submissão ter passado por tais consistências, um arquivo CGI é acionado.

Para cada programa, há um arquivo CGI responsável pela entrada de dados. Assim, atualmente o sistema conta com "gencan.cgi" e "algencan.cgi". A estrutura desses arquivos é a mesma e será pormenorizada a seguir. No entanto, justifica-se criar arquivos de entrada diferentes para programas de otimização diferentes, pois cada programa exige seus próprios arquivos de dados e parâmetros. Na Figura 3.1, é apresentada a estrutura geral dos arquivos que gerenciam o armazenamento de dados e, no restante da seção, é feita uma sucinta descrição de cada componente dessa estrutura.

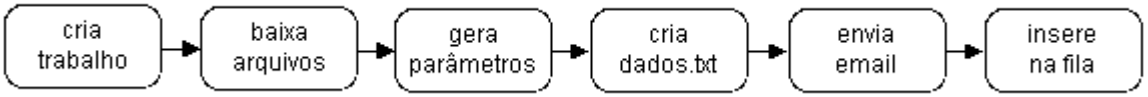



Figura 3.1: Estrutura de "gencan.cgi" e "algencan.cgi".

### 3.1.1 Cria trabalho

O primeiro passo realizado pelo sistema é a geração de um nome para o trabalho submetido pelo usuário. Esse nome tem 16 dígitos, na forma "userYYMMDDHHMMSS", em que YY é o ano corrente, MM o mês corrente e assim sucessivamente até SS, indicando os segundos. O prefixo "user" é comum a todos os trabalhos. Um sufixo do tipo "-i" (i=1,2,..) é colocado quando há um i-ésimo trabalho sendo submetido no mesmo segundo. Assim, por exemplo, suponha que dois trabalhos sejam submetidos às 16:03:42 do dia 27/08/2010. Neste caso, dois trabalhos com os seguintes nomes serão criados: "user100827160342" e "user100827160342-1".

Em seguida, um diretório é criado com o nome do trabalho. Este diretório hospedará todos os arquivos e conterá todas as informações do usuário.

### 3.1.2 Baixa arquivos

Depois de criado o diretório do trabalho, são analisadas as extensões dos arquivos enviados, a fim de se descobrir a linguagem em que cada um foi escrito. No caso do GENCAN e ALGENCAN, são aceitas funções em Fortran, C e C++. Caso o usuário envie uma extensão diferente das esperadas (.f, .for, .c, .cpp, .c++, ...), a submissão é abortada. No ALGENCAN, também é verificado se não foram enviados arquivos de linguagens diferentes, pois nesse programa, todas as funções enviadas tem de estar escritas na mesma linguagem.

Em seguida, os arquivos são salvos na pasta do usuário, com a condição de não superar o tamanho de 1Mb cada um. Caso contrário, a submissão é cancelada.

### 3.1.3 Gera parâmetros

Uma vez gravados os arquivos fornecidos pelo usuário, captura-se os parâmetros selecionados (tipo de derivada, tolerâncias, tipo de saída, etc..) e gera-se o arquivo de parâmetros do programa de otimização. Para o GENCAN, esse arquivo é gerado sempre em Fortran. Para o ALGENCAN, a extensão do arquivo depende da linguagem escolhida.

Se a opção "salvar parâmetros" tiver sido escolhida, então um *cookie* é criado no navegador do usuário. O *cookie* tem como nome "OTIMO_SOLVER", em que SOLVER é o nome do programa de otimização escolhido. Assim, toda vez que o usuário visitar a página do programa de otimização, os parâmetros da última submissão serão recuperados.

### 3.1.4 Cria dados.txt

Na pasta do usuário, é criado um arquivo denominado "dados.txt", com informações tais como o nome do trabalho, a senha, o programa de otimização escolhido, a linguagem das funções enviadas, os tipos de derivadas, etc.. Essas informações serão necessárias em outras etapas do sistema, como na compilação, processamento e geração de resultados. É importante salientar que as cinco primeiras linhas desse arquivo são padronizadas para todos os programas de otimização e contêm, na ordem, o nome do trabalho, senha, programa de otimização, endereço eletrônico (ou vazio, caso o usuário não tenha fornecido) e linguagem escolhida. A partir da sexta linha, o arquivo pode variar entre os programas de otimização.

### 3.1.5 Envia mensagem eletrônica

Um trabalho só passa por esta etapa se o usuário tiver preenchido o campo "email", na página do programa de otimização. Antes da mensagem ser enviada, é feita uma verificação, através do uso de expressões regulares (veja [8]), se o endereço de correio eletrônico está no formato adequado e se não contém nenhum caractere ou comando malicioso.

Em seguida, uma mensagem é enviada ao usuário, indicando que seu problema foi submetido, informando-lhe o nome e a senha do trabalho e alguns *links* para visualização dos resultados finais, cancelamento de trabalho e visualização das filas.

### 3.1.6 Insere na fila

Finalmente, o trabalho é inserido na fila de compilação. Se for o primeiro da fila, o arquivo "compila.pl" é chamado e o problema é compilado. Caso contrário, uma mensagem indicando que o trabalho foi colocado na fila de compilação é exibida.

## 3.2 Compilação

O *script* "compila.pl", chamado pelos arquivos que gerenciam a entrada de dados ("gencan.cgi" e "algencan.cgi"), é responsável pelo gerenciamento da fila de compilação do sistema e será discutido em detalhes na Seção 3.4. Nesta seção, são apresentados os *scripts* "comp_gencan.pl" e "comp_algencan.pl", responsáveis primordialmente pela compilação das funções, e que são chamados pelo *script* "compila.pl".

Cada arquivo "comp_solver.pl" tem como parâmetro de entrada o nome do trabalho e como saída um arquivo executável, pronto para ser rodado com o fim de fornecer a solução do problema enviado. A Figura 3.2 esquematiza a estrutura dos arquivos "comp_solver.pl". Segue abaixo uma breve descrição dos passos por eles executados.

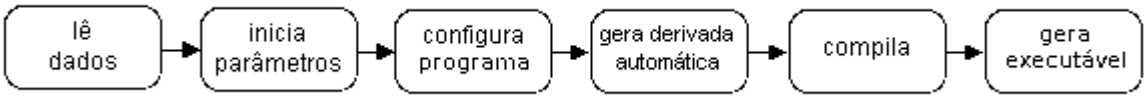



Figura 3.2: Estrutura de "comp_solver.pl".

### 3.2.1 Lê dados

A partir do arquivo de dados guardado na pasta do usuário, obtém-se as informações que serão necessárias nessa etapa, como a linguagem dos arquivos enviados, a dimensão do problema, o tipo de derivadas, etc..

### 3.2.2 Inicia parâmetros

Os programas GENCAN e ALGENCAN exigem o envio da rotina "inip", para a atribuição de valores iniciais a alguns parâmetros (veja pág. 11). No Ótimo, o usuário pode ignorar o envio desse arquivo e fornecer apenas a dimensão e o número de restrições, ou ainda enviar o "inip" apenas com alguns parâmetros e, assim, usar os valores padrão para os parâmetros não fornecidos.

Se o usuário optou por não enviar "inip", um "inip" com os valores padrão (denominado "inip padrão") é criado. Caso o usuário o tenha enviado, é colocado no "inip padrão" uma chamada, no final do código, ao arquivo "inip" enviado pelo usuário. Assim, é possível adotar os valores padrão para os parâmetros não definidos pelo usuário.

### 3.2.3 Configura programa de otimização

Estabelecidos os parâmetros iniciais, há uma chamada ao *script* "config_solver.pl" (ou seja, até aqui, "config_gencan.pl" ou "config_algencan.pl"). Esse *script* é responsável pela configuração dos arquivos dos programa de otimização. No GENCAN e ALGENCAN, o *script* tem a função de:

- Definir a dimensão do problema (no ALGENCAN, também o número de restrições). Assim, todos os vetores que dependem dessa dimensão terão somente o espaço necessário de memória alocada.

- Definir o caminho de saída, para que os arquivos contendo a solução e detalhes das iterações sejam gravados na pasta do usuário.

- Retirar a chamada à função auxiliar para o uso de derivada automática. Em alguns casos, o uso de derivadas automáticas exige a chamada de uma função auxiliar no

início do código dos programas de otimização (veja Seção 3.5.2). Se essa função não for necessária, a linha contendo a chamada é retirada.

### 3.2.4 Gera derivada automática

Nesta etapa, é chamado o *script* "gerader_solver.pl" (ou seja, "gerader_gencan.pl" ou "gerader_algencan.pl"), que retorna três arquivos binários com a extensão ".1", contendo as estruturas para o cálculo das derivadas da função objetivo (GENCAN e ALGENCAN) e mais três arquivos binários com a extensão ".2" contendo as estruturas para o cálculo das derivadas das restrições (ALGENCAN). Tal *script* será detalhado na Seção 3.5.2.

### 3.2.5 Compila

Uma vez completadas com sucesso as etapas acima, torna-se possível a compilação das rotinas para a resolução do problema.

Internamente, os programas GENCAN e ALGENCAN precisam de todas as funções definidas em seus códigos para poderem gerar o executável que resolverá um problema. Por exemplo, suponha que o usuário escolheu a opção de derivada por diferenças finitas no uso do GENCAN. Ainda assim, as funções relativas à primeira e segunda derivadas ("evalg", "evalh", "evalhlp") deverão estar presentes (com o corpo vazio) durante o *link* com as rotinas do programa de otimização. Portanto, há três situações distintas de compilação:

- Compilação das funções enviadas pelo usuário.
- Compilação das funções com o corpo vazio, que não exercem influência mas precisam estar presentes.
- Compilação das funções auxiliares contendo chamadas para as derivadas geradas pelo Adol-C (veja Seção 3.5.2).

Cabe a esta parte do *script* "comp_solver.pl" verificar os tipos de derivadas escolhidas e compilar adequadamente as funções de acordo com as situações listadas acima. Em seguida, são compilados os arquivos dos programas de otimização.

### 3.2.6 Gera executável

Esse trecho do *script* faz a ligação (*link*) entre as funções compiladas no item anterior (as enviadas pelo usuário, auxiliares e do programa de otimização) com a biblioteca do Adol-C e gera o executável "executar", que rodará o programa de otimização.

## 3.3 Execução

Na seção anterior, foi visto que "compila.pl" (responsável pela fila de compilação) chama "comp_solver.pl" a fim de compilar e fazer o *link* dos arquivos, e então gerar o programa executável. Na sequência, ainda em "compila.pl", o trabalho compilado com sucesso é inserido na fila de execução e, então, caso o trabalho seja o primeiro da fila, é chamado o *script* "execut.pl", responsável pelo gerenciamento desta última fila. Um esquema com o funcionamento e a interação entre as filas será mostrado na próxima seção.

A parte de execução propriamente dita, contida em "execut.pl", resume-se a rodar o arquivo executável gerado na fase de compilação (localizado na pasta do usuário e com o nome "executar"). Para tal, é chamado um *shell script*, denominado "roda", que roda o arquivo executável da pasta do usuário. Então, "execut.pl" chama o *script* "gerahtml_solver.pl" que cria a página de saída, contendo a solução encontrada pelo programa ou uma mensagem de erro. Por fim, se o usuário forneceu seu endereço eletrônico, uma mensagem é enviada indicando o término do trabalho, com um atalho para a solução (ou a mensagem de erro) encontrada.

## 3.4 Filas

Com o objetivo de evitar que o servidor do Ótimo seja sobrecarregado, foram criadas filas para organizar os trabalhos submetidos. Existem duas filas, uma para compilação e uma para execução. Essa divisão é justificada, pois o tempo de compilação, em geral, é inferior ao de execução de um problema. Com efeito, imagine a situação de um usuário que tenha de esperar por horas na fila para receber a mensagem indicando que seu problema não compilou. Portanto, em geral, haverá sempre um trabalho sendo compilado e um trabalho sendo executado simultaneamente e independentemente no sistema.

Quando um trabalho é submetido, um *script* do tipo "solver.cgi" salva os dados do usuário e coloca o trabalho na fila de compilação. Se o trabalho não for o primeiro da fila, o usuário recebe uma mensagem indicando o nome e a senha do trabalho, para posterior acesso. Se o trabalho for o primeiro, "compila.pl" é chamado iniciando o laço de compilação. Veja a Figura 3.3.

Quando a compilação desse trabalho termina, verifica-se a existência de outros trabalhos a serem compilados. Assim, o laço irá perdurar até que o último trabalho da fila seja compilado. Voltando ao trabalho submetido, se houver erro na compilação, uma mensagem na tela (e opcionalmente, uma mensagem eletrônica) é enviada e a submissão é cancelada. Caso contrário, o trabalho é inserido na fila de execução. Caso o mesmo seja o primeiro da fila, o laço de execução é iniciado. Veja o fluxo à esquerda da Figura 3.4.

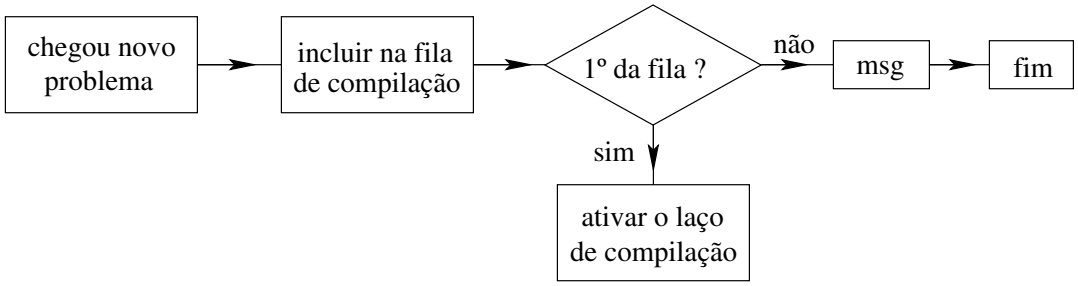



Figura 3.3: Gerenciamento da entrada na fila de compilação em "solver.cgi".

Quando a execução termina, verifica-se a existência de outros trabalhos a serem executados. Novamente, o laço ficará ativo enquando houver trabalhos na fila de execução. Por fim, depois do trabalho ter sido executado, uma página com a solução ou com a mensagem de erro é exibida. Veja o fluxo à direita da Figura 3.4.

## 3.5 Diferenciação automática

Diferenciação automática é um conjunto de técnicas, baseadas na aplicação da regra da cadeia, para obter as derivadas de uma função escrita como um programa de computador. Explora-se o fato de que todo programa de computador, não importando o quão complicado seja, executa uma sequência de operações aritméticas elementares. Ao aplicar a regra da cadeia repetidamente a essas operações, derivadas de qualquer ordem podem ser calculadas automaticamente.

Existem outros métodos para a obtenção das derivadas de uma função, como aproximação numérica por diferenças finitas ou manipulação de expressões algébricas simbólicas (conhecida como diferenciação simbólica). No entanto, a diferenciação automática apresenta vantagens como: eficiência em termos de custo computacional, aplicação em fórmulas de uma linha assim como em fórmulas de mil linhas, possibilidade de ser produzida com esforço humano mínimo e, principalmente, faz o cálculo exato das derivadas (exceto pelo erro de ponto flutuante, naturalmente). Veja em [9] mais informações sobre diferenciação automática, incluindo os programas que usam esse método, livros texto, artigos, etc.

A ferramenta usada pelo Ótimo para o cálculo das derivadas é o Adol-C (Automatic Differentiation by OverLoading in C++), que usa o método de diferenciação automática comentado acima. O pacote Adol-C faz o cálculo das derivadas de primeira ordem e de ordem superior de funções vetoriais definidas em um programa de computador escrito em C ou C++. As rotinas resultantes contendo as derivadas podem ser chamadas por programas em C/C++, Fortran ou outra linguagem qualquer para qual possa ser feito

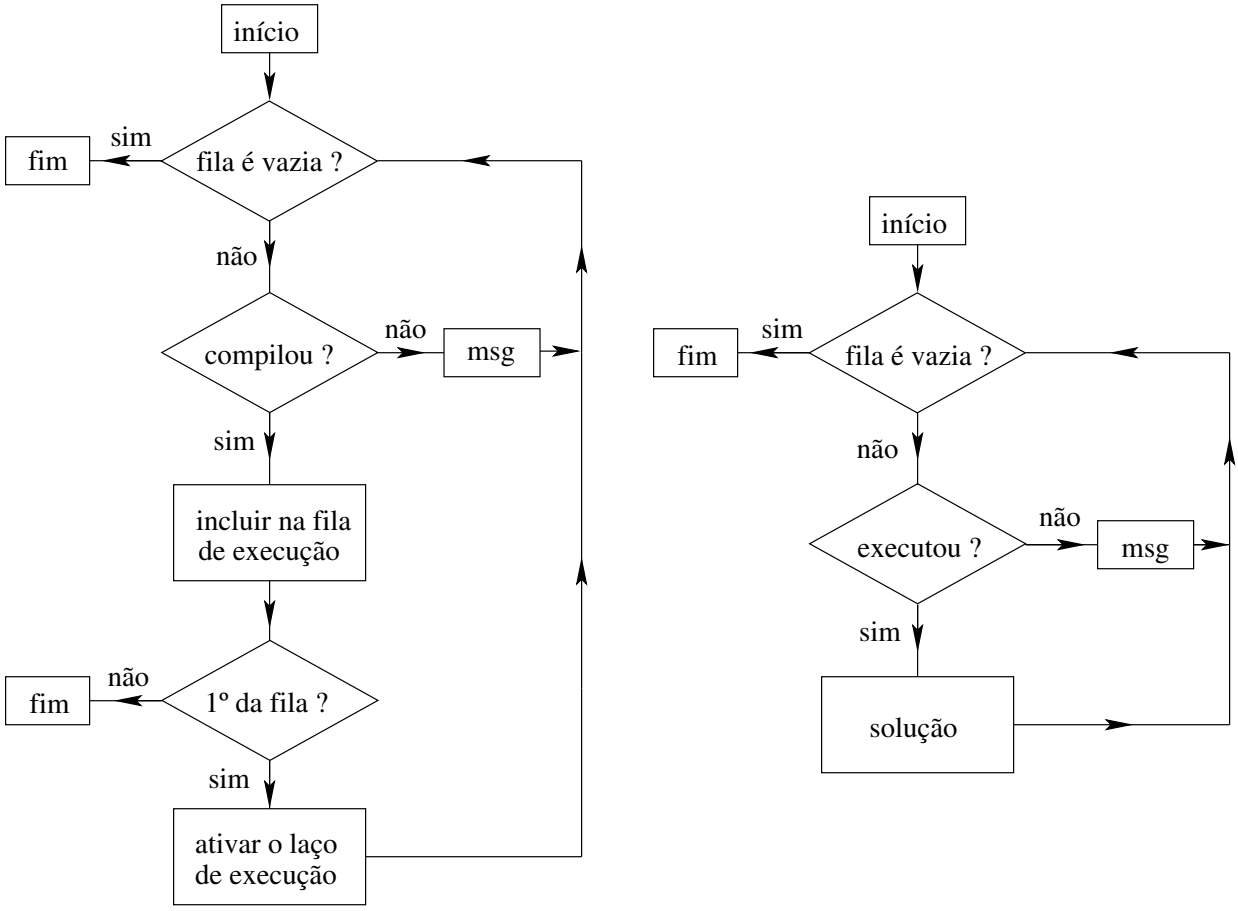



Figura 3.4: Gerenciamento da fila de compilação em "compila.pl" (à esquerda) e da fila de execução em "execut.pl" (à direita).

um *link* com C.

### 3.5.1 Como usar o Adol-C

O Adol-C foi desenvolvido de maneira que poucas modificações no código do usuário são necessárias para a geração das derivadas. A seguir, estão os passos para a geração das derivadas.

- Redeclaração do tipo das variáveis: Todas as variáveis que possam ser consideradas como quantidades diferenciáveis em algum momento durante a execução do programa serão chamadas de "variáveis ativas" e devem ser declaradas como "adouble", que é um tipo escalar do Adol-C cuja parte real é do tipo padrão "double". Assim, a variável independente e todas as variáveis que dela dependam direta ou indiretamente devem ser redeclaradas como "adouble". Para as outras variáveis, pode ser

mantido o tipo padrão "double", "float" ou "int". Mas note que é preciso incluir no código o cabeçalho "adouble.h" para que o compilador entenda esse novo tipo e suas operações.

- Definição da seção de avaliação: A função cuja derivada deseja-se calcular deve estar entre chamadas das funções "trace_on(tag)" e "trace_off(file)" do Adol-C. Estas definem uma "seção ativa", que gravará as derivadas em uma estrutura sequencial chamada "tape". Se o parâmetro binário "file" for 1, essa estrutura é gravada em disco, caso contrário, fica apenas na memória. O parâmetro "tag", que deve ser um inteiro não negativo, serve para identificar a qual função pertence determinada derivada. No momento de avaliar a derivada, tal parâmetro será requerido. Em tempo, as duas funções do Adol-C comentadas acimas têm seus protótipos declarados em "taputil.h", que não precisa ser incluído pois o mesmo já é incluído em "adouble.h".

- Especificação das variáveis dependentes e independentes: Uma ou mais variáveis ativas, lidas ou iniciadas por valores constantes ou por variáveis passivas (uma variável não ativa é uma variável passiva), devem ser definidas como variáveis independentes. Para uma variável $x$ ser definida como independente, é necessária uma atribuição da forma

  $$x <<= px,$$

  tal que px é uma variável de um tipo passivo. Assim, é atribuido ao campo real de $x$ o valor de $px$. Esse comando deve preceder qualquer atribuição a $x$.

  Analogamente, para um variável $y$ ser definida como dependente, é necessária uma atribuição da forma

  $$y >>= py,$$

  tal que $py$ é uma variável do tipo passivo. Assim, é atribuído a $py$ o valor do campo real de $y$. Esse comando não deve ser sucedido por atribuições a $y$.

- Recompilação do código e *link* com a biblioteca do Adol-C: Uma vez feitas as inserções e modificações do código descritas acima, basta compilá-lo em C++ e então ligá-lo com a biblioteca do Adol-C, armazenado no arquivo "libadolc.a", para gerar o executável. Se o usuário optou por salvar as derivadas em disco (ver parâmetro "file", acima), ao rodar o executável, três arquivos binários contendo as derivadas são criados. Os nomes desses arquivos contêm, em suas extensões, o número do parâmetro "tag", comentado acima.

Uma vez que as derivadas foram geradas, a preocupação agora é como calculá-las. O Adol-C fornece funções que facilitam a avaliação das derivadas. Para tal, é assumido que, depois da execução de uma seção ativa, esteja na memória (ou em disco) o "tape" com seu identificador ("tag") contendo um registro detalhado do processo computacional pelo qual

os valores finais, $y$, das variáveis dependentes foram obtidos a partir dos valores iniciais, $x$, das variáveis independentes.

Seja esta relação denotada por

$$F\colon \mathbb{R}^n \to \mathbb{R}^m, \qquad x \to F(x) \equiv y.$$

Seria possível, por exemplo, calcular a matriz $F'(x)$ usando-se a seguinte função do Adol-C:

```
int jacobian(tag,m,n,x,J)
```

Cujos parâmetros são:

```
short int tag;                          // tape de identificação
int m;                                  // número de variáveis dependentes
int n;                                  // número de variáveis independentes
double x[n];                            // vetor independente x
double J[m][n];                         // matriz Jacobiana resultante F'(x)
```

O Adol-C conta ainda com funções para o cálculo de $\nabla F(x)$, $\nabla^2 F(x)$, $[\nabla^2 F(x)]v$, $[F''(x)]v$, $\nabla^2 F(x)$ esparsa, $F''(x)$ esparsa, entre outros. No apêndice 5.4, há uma lista das funções do Adol-C usadas pelo Ótimo. Note que essas funções estão escritas em C, ou seja, podem ser chamadas por programas escritos em qualquer linguagem que possa ser ligada ao C, como Fortran, por exemplo. Para usar essas funções, o usuário deve incluir o cabeçalho "drivers.h".

Segue abaixo um exemplo simples do uso do Adol-C para o cálculo do gradiente da função $f(x) = x_1^2 + x_2^2$ , escrita como um programa em C++ :

```
#include <adouble.h>              // Uso to tipo adouble e geração do tape
#include <drivers/drivers.h>      // Uso do driver "gradient"
main()
{
  /* Inicalização das variáveis */
  int n=2;
  double xp[n],yp;
  xp[0]=1; xp[1]=2;              // valor arbitrário para xp
  yp=0;

  /* Definição das variáveis e do identificador (geração do tape) */
  trace_on(1);                   // tag=1
  adouble x[n];
```

```
  adouble y=0;
  for(int i=0;i<n;i++){
    x[i] <<= xp[i];              // definindo variáveis independentes
  }
  for(int i=0;i<n;i++){
    y = y + x[i] * x[i];
  }
  y >>= yp;                      // definindo variável dependente
  trace_off();                   // file=0 (valor padrão)

  /* Avaliação do gradiente */
  double g[n];
  gradient(1,n,xp,g);           // chamada ao driver "gradient" com tag=1

  /* Impressão do gradiente */
  printf(`` g[0] = %f \n g[1] = %f \n'',g[0],g[1]);
}
```

Como no Ótimo o Adol-C está instalado na pasta "/home/otimo/adolc_base", o código acima ("teste.cpp") poderia ser compilado com o comando:

```
g++ -I/home/otimo/adolc_base/include/adolc teste.cpp
/home/otimo/adolc_base/lib/libadolc.a -o executavel;
./executavel;
```

### 3.5.2 O Adol-C no Ótimo

#### Geração das derivadas

O *script* "gerader_solver.pl", chamado por "comp_solver.pl", é o responsável por gerar as derivadas no Ótimo. Ou seja, ao fim de sua execução, são gravados, na pasta do usuário, os arquivos gerados pelo Adol-C. No GENCAN, existem três arquivos com a extensão ".1", relacionados às derivadas de "evalf" (função que avalia a função objetivo). No ALGENCAN, além desses três, existem outros três com a extensão ".2", relacionados às derivadas de "evalc" (função que avalia as restrições). Nos próximos parágrafos, será descrito como "gerader_solver.pl" gera esses arquivos para evalf e para evalc.

Primeiramente, é verificada a linguagem do arquivo "evalf" enviado. Deve-se lembrar que o Adol-C só é capaz de gerar as derivadas para funções em C++. Assim, caso "evalf" esteja em Fortran, o programa "f2c" (*Fortran to c*) é usado para convertê-la para C++. E, caso "evalf" esteja em C, um cabeçalho é acrescentado à função.

Na sequência, o *script* "converte_f.pl" é chamado. Esse *script* gera o arquivo "avalia_f_autodif.cpp", que contém a função "evalf" com os tipos redeclarados para o uso do Adol-C.

O sistema conta ainda com dois arquivos auxiliares em C++ para a geração das derivadas: "main_X.cpp" ("X" depende da linguagem de inip: X = "f" ou "c" ou "cpp") e "aux_Y.cpp" ("Y" depende da linguagem de evalf: Y = "f" ou "c" ou "cpp").

O arquivo "main_X.cpp" contém a função "main", que declara as variáveis, inicia "x" (através da chamada a "inip", motivo pelo qual é necessário um arquivo "main" para cada linguagem de "inip") e chama a função "eval" (definida em "aux_Y.cpp").

O arquivo "aux_Y.cpp" contém a função "eval", que é responsável pela definição da seção de avaliação e pela especificação das variáveis dependentes e independentes, e que contém uma chamada para "evalf", depois de modificada por "converte_f.pl". Veja abaixo um exemplo de arquivo "aux_cpp.cpp":

```
// Caso em que evalf está em C++
#include <adolc/adouble.h>
void evalf(int *n,adouble *x,adouble *y,int *flag);

void eval(int *n,double *px,double *py,int *flag)
{
 int m=1;
 short int tag=1;          // os tapes terão a extensão ".1"
 short int file=1;
 trace_on(tag);
 adouble *x,*y;
 x = new adouble[*n];
 y = new adouble[m];
 for (int i=0;i<*n;i++){
   x[i]<<=px[i];
 }
 evalf(n,x,y,flag);
 for(int i=0; i<m; i++)
   y[i]>>=py[i];
 delete[] y;
 delete[] x;
 trace_off(file);
}
```

Para gerar os arquivos com as derivadas, basta compilar todas as funções acima mencionadas e ligá-las às bibliotecas do Adol-C e "f2c", através dos comandos:

```
g++ main_X.cpp aux_Y.cpp inip.cpp avalia_f_autodif.cpp libf2c.a libadolc.a
-o executavel;
./executavel;
```

No entanto, cabe ressaltar que, no sistema, essa compilação é feita diretamente com os objetos "main_X".o e "aux_Y.o", gerados uma única vez.

O cálculo dos arquivos com as derivadas para "evalc" (para o ALGENCAN) é completamente análogo ao descrito acima para "evalf", exceto por um detalhe na função "aux_Y.cpp".

A função "evalc" avalia as restrições do problema, recebendo o índice da i-ésima restrição como argumento. Assim, "evalc" é uma função da forma: $F_i : \mathbb{R}^n \to \mathbb{R}$ , enquanto o Adol-C contém funções que avaliam as derivadas de funções vetoriais $F : \mathbb{R}^n \to \mathbb{R}^m$. Logo, é necessária uma adaptação na geração e na avaliação das derivadas a fim de tornar compatíveis os dois programas. Veja abaixo o arquivo adaptado "aux_Y.cpp".

```
#include <adolc/adouble.h>
void evalc(int n,adouble *x,int k,adouble *y,int *flag);

void eval(int *n,int *m,double *px,double *py,int *flag)
{
 int tag=2;                    // os tapes terão a extensão ".2"
 int file=1;
 trace_on(tag);
 adouble *x,*y;
 x = new adouble[*n];
 y = new adouble[*m];
 for (int i=0;i<*n;i++){
   x[i]<<=px[i];
 }
 for(int j=1;j<=*m;j++){
   evalc(*n,x,j,&y[j-1],flag); // adaptação para a representação do Adol-C
 }
 for(int i=0; i<*m; i++)
   y[i]>>=py[i];
 delete[] y;
 delete[] x;
 trace_off(file);
```

```
}
```

No algoritmo acima, perceba que "evalc" é chamado dentro de um laço, de modo que cada componente do vetor da variável dependente y é o resultado da avaliação da i-ésima restrição sobre a variável independente x. Assim, os "tapes" (estrutura com as derivadas) são gerados sobre um "evalc" transformado para uma função vetorial $F: \mathbb{R}^n \to \mathbb{R}^m$.

### Avaliação das derivadas

No GENCAN, o gradiente da função objetivo deve ser fornecido usando-se a função "evalg", e a Hessiana da função objetivo através de "evalh" ou "evalhlp" (produto da Hessiana da função objetivo por um vetor de entrada). Assim, o Ótimo conta com funções que avaliam as derivadas geradas pelo Adol-C para essas três situações. Por uma escolha arbitrária, para o GENCAN essas funções foram escritas em C.

A função "evalg", que faz parte do arquivo "avalia_gder.c", é mostrada abaixo:

```
void evalg(int *n,double *x,double *g,int *flag){
  int tag = 1;
  gradient(tag,*n,x,g);
  *flag=0;
}
```

A função acima faz uso de uma função do Adol-C, "gradient", que retorna o gradiente da função avaliado no ponto x. Lembre-se que todas as derivadas de "evalf" estão associadas a $tag = 1$.

A função "evalhlp" faz uso de uma função do Adol-C, "hess_vec" (veja apêndice 5.4), que retorna a Hessiana da função multiplicada por um vetor p, como mostrado abaixo:

```
void evalhlp(int *n,double *x,int *m,double *lambda,double *p,double *hp,
int *goth,int *flag){
  int tag = 1;
  hess_vec(tag,*n,x,p,hp);
  *flag=0;
}
```

Por fim, a função "evalh" faz parte do arquivo "avalia_hder.c", mostrado a seguir.

```
\* Variáveis globais *\
int nnz;
```

```
unsigned int* rind=NULL;
unsigned int* cind=NULL;
double* values=NULL;

\* Função auxiliar, para o cálculo do padrão de esparsidade *\
void esparsid_(int *n,double *x){
  int tag=1;
  int repeat=0;        // calcula padrão de esparsidade
  sparse_hess(tag,*n,repeat,x,&nnz,&rind,&cind,&values);
}

\* Função evalh *\
void evalh(int* n,double *x,int *hlin,int *hcol,double *hval,int *nnzh,
int *flag){
   *flag = 0;
   int repeat=1;      // usa padrão anterior
   int i,tag;
   tag=1;
   sparse_hess(tag,*n,repeat,x,&nnz,&rind,&cind,&values);
   *nnzh=nnz;
   for(i=0;i<*nnzh;i++){           // Rearranjando os índices
     hlin[i]=rind[i]+1;
     hcol[i]=cind[i]+1;
     hval[i]=values[i];
   }
}
```

No arquivo mostrado acima, além da função "evalh" para o cálculo da Hessiana, há também a função "esparsid" e mais quatro variáveis globais. Para entender o porquê deste código ser mais complexo que os outros, deve-se analisar a função "sparse_hess" (veja o Apêndice 5.4). Ao rodar "sparse_hess" com o parâmetro repeat=0, o Adol-C calcula o padrão de esparsidade da Hessiana e, então, os valores de cada elemento. Se "sparse_hess" for chamado com repeat=1, o padrão de esparsidade é aproveitado e apenas os valores dos elementos são calculados.

Dessa maneira, com o objetivo de economizar tempo de processamento, o sistema faz uma única chamada à função esparsid antes da chamada do programa de otimização (no GENCAN, a chamada é feita no arquivo "algencanma.f") e o padrão de esparsidade é guardado nas variáveis globais "nnz", "rind" e "cind". Assim, quando o programa de otimização faz chamadas à função "evalh", a mesma chama "sparse_hess" usando o padrão de esparsidade já calculado.

O laço no fim do código de "evalh" adequa os índices da matriz esparsa, pois, no GENCAN, os índices das linhas e colunas começam em um e, no Adol-C, começam em zero.

No ALGENCAN, as primeiras derivadas devem ser fornecidas em "evalg" (função objetivo) e "evaljac" (restrições) ou em "evalgjac" (função objetivo e restrições) . As segundas derivadas são fornecidas através de "evalh" (Hessiana da função objetivo) e "evalhc" (Hessiana das restrições), ou através de "evalhlp" (Hessiana do Lagrangiano multiplicada por um vetor de entrada), ou ainda, através de "evalhl" (Hessiana do Lagrangiano). O Ótimo conta com funções que avaliam as derivadas geradas pelo Adol-C para serem usadas por "evalg", "evaljac" e "evalhlp".

A função "evalg" é fornecida pelo mesmo arquivo "avalia_gder.cpp", exibido na pág. 50, para o GENCAN. Para a função "evaljac", existe o problema de compatibilidade entre o Adol-C e o ALGENCAN, já comentado na pág 49. A função evaljac tem como entrada a i-ésima restrição, e como saída um vetor esparso com a derivada dessa restrição. Já o Adol-C pode calcular a derivada de uma função vetorial em que a i-ésima componente representa a i-ésima restrição, retornando a derivada na representação esparsa, na forma matricial. O arquivo "avalia_jacder.cpp" mostrado abaixo faz essa conversão:

```
\* Variáveis globais *\
int nnz;
unsigned int* rind=NULL;
unsigned int* cind=NULL;
double* values=NULL;
int* posl=NULL;
int maux;

\* Calcula e guarda o padrão de esparsidade *\
void esparsid(int *n,double *x,int *m){
  int tag=2;
  int repeat=0;
  maux=*m;
  sparse_jac(tag,*m,*n,repeat,x,&nnz,&rind,&cind,&values);

  \* Muda a representação do padrão de esparsidade *\
  int nlinhas=rind[nnz-1]+1;
  posl=new int[nlinhas+1];
  int linha=1;
  posl[0]=1;
  for(int i=0;i<nnz;i++){
    if(rind[i]+1!=linha){
```

```
      linha=rind[i]+1;
      posl[linha-1]=i+1;
    }
  }
  posl[linha]=nnz+1;
}

\* Função evaljac *\
void evaljac(int n,double *x,int ind,int *indjac,double *valjac,
int *nnzjac,int *flag){
  *flag = 0;
  int tag=2;
  int repeat=1;
  int m=maux;
  sparse_jac(tag,m,n,repeat,x,&nnz,&rind,&cind,&values);

  \* Converte para a representação do ALGENCAN *\
  *nnzjac=posl[ind]-posl[ind-1];
  for(int i=0;i<*nnzjac;i++){
    indjac[i]=cind[i+posl[ind-1]-1];
    valjac[i]=values[i+posl[ind-1]-1];
   }
}
```

Antes da execução do programa de otimização, a função "esparsid" é chamada. Então, o padrão de esparsidade é calculado (pela chamada à "sparse_jac" com repeat=0) e armazenado no vetor "posl", que é uma variável global. Durante a execução do programa, sempre que "evaljac" é chamado, este usa o padrão de esparsidade em "posl" e, através de uma chamada a "sparse_jac", calcula o valor do jacobiano no ponto dado. Finalmente, é feita a conversão do jacobiano para o padrão do ALGENCAN. Veja os parâmetros da função "sparse_jac", do Adol-C, no apêndice 5.4.

Por fim, a função "evalhlp" é mostrada abaixo:

```
void evalhlp(int n,double *x,int m,double *lambda,double sf,double *sc,
double *p,double *hp,int *goth,int *flag) {
  *flag=0;
  double* z = new double[n];           \\ vetor auxiliar temporário
  hess_vec(1,n,x,p,z);                 \\ Hessiana de evalf
  lagra_hess_vec(2,m,n,x,p,lambda,z); \\ Hessiana de evalc

  \* Hessiana do Lagrangiano *\
```

```
  for(int i=0;i<n;i++){                    \\ Hessiana do Lagrangiano
    hp[i]=hp[i]+z[i];
  }
}
```

Na rotina acima, foram usadas as funções do Adol-C "hess_vec", para o cálculo do produto da Hessiana da função objetivo por um vetor, e "lagra_hess_vec", que calcula o produto da Hessiana das restrições por um vetor e, em seguida, o produto do vetor resultante pelo vetor de multiplicadores de Lagrange. Somando os dois vetores resultantes da chamada destas funções, obtém-se o produto da Hessiana do Lagrangiano por um vetor.

### 3.5.3 Testes com o Adol-C

Nas Seções 3.1 e 3.2, mencionou-se que o usuário tem três opções para o cálculo das derivadas de primeira e segunda ordem nos programas GENCAN e ALGENCAN. O usuário pode fornecer o arquivo com a derivada, usar diferenças finitas ou usar diferenciação automática através do pacote Adol-C. Nesta seção, alguns testes são realizados trocando a forma de cálculo das derivadas, com o propósito de avaliar a eficiência do Adol-C. A versão do Adol-C utilizada foi a 2.1.8, associada ao pacote ColPack, empregado na geração de derivadas que envolvem matrizes esparsas.

No primeiro teste, submeteu-se ao GENCAN, o problema irrestrito

$$\min \quad f(x) = \sum_{i=1}^{n} (x_i - 3)^2,$$

cujo valor ótimo é zero. Na Tabela 3.1 são apresentados os resultados alcançados para alguns valores de $n$. Nota-se que, até que fosse atingido o limite de memória, a diferenciação automática foi tão eficiente quanto enviar o arquivo com as derivadas. Por outro lado, o método de diferenças finitas já se mostra ineficiente para $n = 300.000$.

O segundo teste consistiu em submeter ao GENCAN o problema

$$\begin{aligned} \min \quad & f(x) = \sum_{i=1}^{n-1} [(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2] \\ \text{suj. a} \quad & -5 \leq x_i \leq 10, \quad i = 1, \ldots, n. \end{aligned}$$

Também nesse caso, o valor ótimo da função objetivo é zero. A Tabela 3.2 contém os resultados alcançados. Mais uma vez, a diferenciação automática se mostrou razoavelmente eficiente, apesar de não ter sido possível gerar derivadas para $n$ igual ou superior a 5.000.

| n | Derivada primeira | Derivada segunda | Tempo de proces.(s) | Número de Iterações | Valor da solução |
|---|---|---|---|---|---|
| 10.000 | fornecida | fornecida | 0,05 | 1 | 2,42e-21 |
| | fornecida | dif. finitas | 0,06 | 1 | 2,42e-21 |
| | fornecida | automática | 0,07 | 1 | 2,42e-21 |
| | dif. finitas | dif. finitas | 2,35 | 3 | 7,88e-27 |
| | automática | automática | 0,08 | 1 | 2,42e-21 |
| 300.000 | fornecida | fornecida | 1,6 | 1 | 3,15e-16 |
| | fornecida | dif. finitas | 1,65 | 1 | 3,15e-16 |
| | fornecida | automática | 1,61 | 1 | 3,15e-16 |
| | dif. finitas | dif. finitas | >600 | - | - |
| | automática | automática | 1,75 | 1 | 3,15e-16 |
| 600.000 | Erro no Adol-C | | | | |

Tabela 3.1: Resultados do problema 1.

| n | Derivada primeira | Derivada segunda | Tempo de proces.(s) | Número de Iterações | Valor da solução |
|---|---|---|---|---|---|
| 1.000 | fornecida | dif. finitas | 0,12 | 36 | 3,33e-20 |
| | fornecida | automática | 0,62 | 35 | 3,12e-20 |
| | dif. finitas | dif. finitas | 3,82 | 35 | 2,36e-16 |
| | automática | automática | 0,58 | 36 | 2,77e-20 |
| | automática | dif. finitas | 0,3 | 35 | 9,69e-20 |
| 3.000 | fornecida | dif. finitas | 0,17 | 33 | 3,67e-20 |
| | fornecida | automática | 1,54 | 32 | 2,62e-20 |
| | dif. finitas | dif. finitas | >600 | - | - |
| | automática | automática | 1,58 | 32 | 5,72e-20 |
| | automática | dif. finitas | 0,77 | 33 | 5,45e-20 |
| 5.000 | Erro no Adol-C | | | | |

Tabela 3.2: Resultados do problema 2.

Por fim, no terceiro teste, submeteu-se ao ALGENCAN o clássico problema de maximizar a distância mínima entre quaisquer dois pontos, em um conjunto com "np" pontos sobre uma "esfera" de dimensão "ndim" (Hard spheres problem). Após algumas manipulações algébricas, esse problema pode ser escrito como

$$
\begin{array}{lll}
\min & z & \\
\text{suj. a} & z \geq < x_i, x_j > & \forall (i,j) \in \{(1..np, 1..np) : i \neq j\} \\
 & \|x_i\|_2^2 = 1, & i = 1, \ldots, np
\end{array}
$$

Da maneira como foi modelado, o problema possui função objetivo linear e restrições não lineares. Naturalmente, a dimensão do problema e o número de restrições são funções de "np" e "ndim". Analisando os resultados na Tabela 3.3, nota-se que a diferenciação automática já não tem bom desempenho para valores relativamente baixos de "n" e "m", devido à complexidade das restrições do problema.

| Dimensões | Derivada primeira | Derivada segunda | Tempo de proces.(s) | Número de Iterações | Valor da solução |
|---|---|---|---|---|---|
| | fornecida | fornecida | 0,1 | 31 | -5,26e-02 |
| n=401 | fornecida | dif. finitas | 0,05 | 31 | -5,26e-02 |
| m=210 | fornecida | automática | 0,58 | 21 | -5,26e-02 |
| ndim=20 | dif. finitas | dif. finitas | 1,19 | 31 | -5,26e-02 |
| np=20 | automática | automática | 0,78 | 18 | -5,26e-02 |
| | automática | dif. finitas | 2,56 | 127 | -5,26e-02 |
| | fornecida | fornecida | 0,22 | 25 | -2,50e-01 |
| n=1.001 | fornecida | dif. finitas | 0,14 | 40 | -2,50e-01 |
| m=15 | fornecida | automática | 34,4 | 2369 | -2,49e-01 |
| ndim=200 | dif. finitas | dif. finitas | 1,32 | 43 | -2,49e-01 |
| np=5 | automática | automática | 24,3 | 11010 | -1,01e+02 |
| | automática | dif. finitas | 6,18 | 98 | -2,50e-01 |
| n=2.501 m=1.275 ndim=500 np=50 | Erro no Adol-C | | | | |

Tabela 3.3: Resultados do problema 3.

## 3.6 Visualização dos resultados

Quando um trabalho é finalizado, uma página HTML denominada "saida.html" é gerada na pasta do usuário. Esta página pode conter a solução encontrada pelo programa de otimização ou uma mensagem de erro. A mensagem de erro pode ter sido gerada na fase

de execução (caso o erro tenha ocorrido durante a execução do programa de otimização) ou na fase de compilação (caso o erro tenha ocorrido durante a compilação das funções enviadas, na geração das derivadas, etc). Nesta seção, será descrito brevemente o *script* "gerahtml_solver.pl" (em que *solver* é o nome do programa de otimização escolhido), responsável pela criação de "saida.html".

Entretanto, antes da descrição de "gerahtml_solver.pl", é importante comentar como são tratadas as mensagens de erro no sistema. Sempre que um programa externo é utilizado (programas de otimização, compiladores, Adol-C, f2c, ...), uma variável guarda a saída de erro padrão do programa. Assim, durante os estágios de compilação e execução, é verificado se essa variável contém alguma informação. Em caso afirmativo, o processamento do trabalho é interrompido e o arquivo "erromsg.txt", contendo os dados dessa variável, é criado na pasta do usuário. Também é guardado em "erromsg.txt" o estágio no qual o trabalho foi cancelado (compilação ou execução).

O *script* "gerahtml_solver.pl" pode ser invocado por "compila.pl" ou "execut.pl". Em "compila.pl", ele é chamado quando ocorre erro na fase de compilação, ou seja, quando a variável que recebe a saída de erro na chamada à "comp_solver.pl" é não vazia. Nesse caso, a chamada a "gerahtml_solver.pl" é feita passando como parâmetro o texto "erro_co" (erro na compilação) e o *script* simplesmente copia o conteúdo de "erromsg.txt" para "saida.html".

Já em "execut.pl", "gerahtml_solver.pl" pode ser chamado quando ocorre um erro na fase de execução, ou seja, quando a variável que recebe a saída de erro na chamada ao *shell script* "roda" é não vazia. Neste caso, analogamente ao anterior, a chamada a "gerahtml_solver.pl" é feita passando como parâmetro o texto "erro_ex" (erro na execução) e o *script* copia o conteúdo de "erromsg.txt" para "saida.html". Caso nenhum erro de execução ou compilação tenha sido encontrado, "execut.pl" invoca "gerahtml_solver.pl" com o parâmetro "ok". Neste caso, "gerahtml_solver.pl" gera uma página contendo as informações básicas geradas pelo programa de otimização (número de iterações, tempo de processamento, etc..), um botão para acesso à solução propriamente dita e opcionalmente, um botão para acesso às informações detalhadas das iterações. Naturalmente, o botão com as informações detalhadas aparecerá somente se o programa de otimização escolhido disponibilizar tal informação e se o usuário tiver optado, na página de configuração dos parâmetros, por ver a saída detalhada .

O botão que permite a visualização da solução encontrada executa o *script* "soluc.cgi", que simplesmente exibe o arquivo gerado pelo otimizador. Analogamente, o botão para visualização da saída detalhada executa "detail.cgi" que também exibe um arquivo de saída do otimizador. No próximo capítulo, mostrar-se-á um exemplo de submissão de um problema e serão exibidas algumas páginas de saída.

## 3.7 Restabelecimento do Ótimo

Por motivos variados, o servidor do sistema pode parar de funcionar, interrompendo o processamento de algum trabalho. O *script* "cron.pl" foi desenvolvido com o intuito de restabelecer o processamento dos trabalhos no Ótimo. Esse *script* verifica se há trabalho na fila de compilação ou execução e se há, associado a esse trabalho, um processo no sistema referente à sua compilação ou execução. Caso haja algum trabalho na fila, mas não exista um processo a ele associado, "cron.pl" reinicia o estágio de compilação ou execução, rodando "compila.pl" ou "execut.pl", respectivamente. Definiu-se que "cron.pl" deve ser rodado de dez em dez minutos (usa-se a função para agendamento de tarefas "crontab", do UNIX), mas é possível alterar esse intervalo modificando o arquivo "/home/otimo/help/tarefas". Para mais detalhes sobre o uso do "crontab" no UNIX, veja [10].

## 3.8 Cancelamento de trabalho

O usuário pode cancelar um trabalho acessando a página de cancelamento. Para tanto, é preciso informar o nome e a senha do trabalho, fornecidos no momento da submissão. Se o trabalho a ser cancelado ainda estiver em alguma fila, aguardando a sua vez, o mesmo simplesmente é retirado da fila. Caso o trabalho já esteja em processamento, o processo referente à ele é finalizado através do comando "kill" do UNIX. O *script* "cancela.cgi" é o responsável por esse cancelamento.

# Capítulo 4

# Exemplo de submissão de um problema

Neste capítulo, será mostrado, através de um exemplo, como resolver um problema no Ótimo. Também serão exibidas as páginas do sistema com que o usuário terá contato durante o processamento de seu trabalho.

Suponha que o usuário deseje resolver o problema

$$\begin{aligned} \min \quad & f(x) = -(x_1^2 + x_2^2) \\ \text{suj. a} \quad & x_1^2 - x_2 = 0 \\ & x_1 + x_2 - 1 \leq 0 \\ & 0 \leq x_1 \leq 1 \\ & 0 \leq x_2 \leq 1. \end{aligned}$$

A região viável deste problema está destacada na Figura 4.1, sendo composta pelos pontos da parábola compreendidos entre a origem e a interseção da parábola com a reta, no primeiro quadrante. As circunferências tracejadas representam as curvas de nível da função objetivo, sendo que o valor da função objetivo diminui conforme o raio das circunferências aumentam. Assim, graficamente nota-se que a solução desse problema é justamente o ponto de interseção da parábola com a reta.

A solução do problema é:

$$x_1 = \frac{-1 + \sqrt{5}}{2} \simeq 0.618, \qquad x_2 = x_1^2 \simeq 0.382.$$

O primeiro passo para resolver esse problema no Ótimo é identificar o programa de otimização mais apropriado para o problema em questão. Nesse caso, por tratar-se de um problema de programação não linear com restrições de igualdade e desigualdade, deve-se escolher o ALGENCAN.

Na página do ALGENCAN, encontram-se as instruções para o envio dos arquivos e definição das funções (veja pág. 18). De acordo com essas especificações, para resolver esse problema, o usuário deverá enviar no mínimo três arquivos contendo, cada um, uma das funções abaixo:

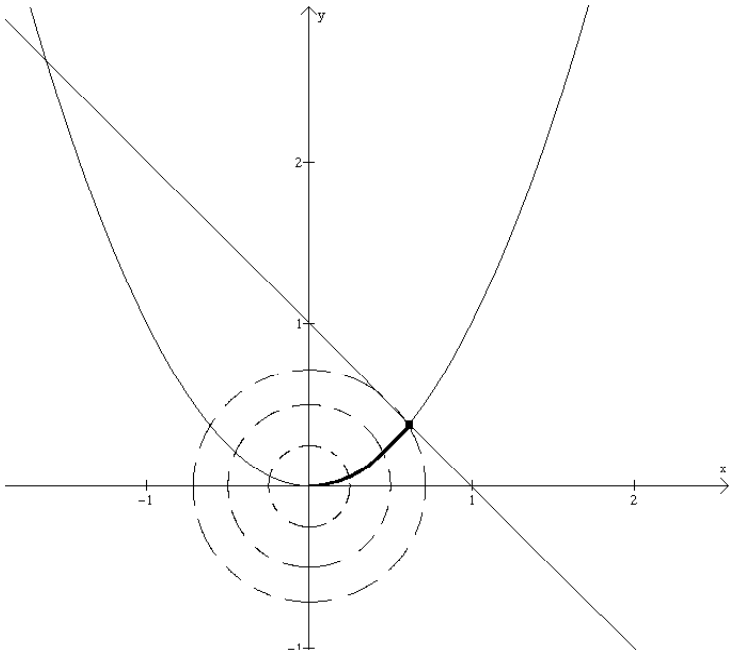



Figura 4.1: A região viável é a linha mais escura e as curvas de nível da função objetivo são as linhas tracejadas.

- inip, que define parâmetros iniciais, limitantes e outras características.
- evalf, que calcula a função objetivo.
- evalc, que calcula as restrições.

Um exemplo da função "inip", em Fortran, é dado abaixo:

```
subroutine inip(n,x,l,u,m,lambda,equatn,linear)
integer m,n
logical equatn(*),linear(*)
double precision l(*),lambda(*),u(*),x(*)
integer i

n = 2
do i = 1,n
  x(i) = 0.5
  l(i) = 0.0
  u(i) = 1.0
end do
```

```
m = 2
do i = 1,m
    lambda(i) =  0.0
end do

equatn(1) = .true.
equatn(2) = .false.

linear(1) = .false.
linear(2) =  .true.

end
```

A subrotina "evalf" pode ser definida como:

```
subroutine evalf(n,x,f,flag)
integer flag,n
double precision f,x(n)

f = - x(1)**2 - x(2)**2

flag = 0

end
```

Note que, caso o objetivo fosse de maximização ao invés de minimização, seria necessário a troca do sinal da função objetivo, já que o ALGENCAN tem como padrão resolver problemas de minimização.

A subrotina "evalc" é dada por:

```
subroutine evalc(n,x,ind,c,flag)
integer ind,flag,n
double precision c,x(n)

if ( ind .eq. 1 ) then
    c = x(1) ** 2 - x(2)

else if ( ind .eq. 2 ) then
    c = x(1) + x(2) - 1
end if
```

```
flag = 0
return

end
```

Observa-se que seria necessário a troca do sinal das restrições de desigualdade caso o operador fosse do tipo "maior que", já que o padrão do ALGENCAN é o operador "menor que".

Com os três arquivos acima, o usuário já poderia enviar o seu problema ao Ótimo. Bastaria escolher, como opção para as derivadas, o método das diferenças finitas ou a diferenciação automática.

Entretanto, neste exemplo, como as derivadas são fáceis de calcular, também serão enviadas as seguintes funções:

- evalg, que calcula o gradiente da função objetivo.
- evaljac, que calcula a Jacobiana das restrições.
- evalh, que calcula a Hessiana da função objetivo.
- evalhc, que calcula a Hessiana das restrições.

Segue abaixo um exemplo de código em Fortran para cada função acima.

A subrotina "evalg" pode ser escrita da seguinte forma:

```
subroutine evalg(n,x,g,flag)

integer flag,n
double precision g(n),x(n)
integer i

do i = 1,n
    g(i) = -2 * x(i)
end do

flag = 0

end
```

A subrotina "evaljac" pode ser codificada como:

```
subroutine evaljac(n,x,ind,jcvar,jcval,jcnnz,flag)

integer flag,ind,n,jcnnz,jcvar(n)
double precision x(n),jcval(n)

if ( ind .eq. 1 ) then

    jcnnz = 2

    jcvar(1) = 1
    jcval(1) = 2 * x(1)
    jcvar(2) = 2
    jcval(2) = - 1

else if ( ind .eq. 2 ) then

    jcnnz = 2

    jcvar(1) = 1
    jcval(1) = 1
    jcvar(2) = 2
    jcval(2) = 1

end if

flag = 0
return

end
```

A subrotina "evalh" seria definida como:

```
subroutine evalh(n,x,hlin,hcol,hval,hnnz,flag)

integer flag,n,hnnz,hcol(*),hlin(*)
double precision hval(*),x(n)

hnnz = 2

hlin(1) = 1
hcol(1) = 1
hval(1) = -2
```

```
hlin(2) = 2
hcol(2) = 2
hval(2) = -2

flag = 0

end
```

Já a codificação de "evalhc" poderia ser:

```
subroutine evalhc(n,x,ind,hclin,hccol,hcval,hcnnz,flag)

integer flag,ind,n,hcnnz,hccol(*),hclin(*)
double precision hcval(*),x(n)

if ( ind .eq. 1 ) then

    hcnnz = 1

    hclin(1) = 1
    hccol(1) = 1
    hcval(1) = 2

else if ( ind .eq. 2 ) then

    hcnnz = 0

end if

flag = 0
return

end
```

Após fornecer os arquivos, o usuário deve escolher o valor dos parâmetros do ALGENCAN, na própria página do programa. Neste exemplo, serão mantidos os valores padrão. É importante ressaltar que, para rodar este caso, foi usado a versão 1.0 do ALGENCAN, disponível no Ótimo. A versão 2.2 do ALGENCAN, também disponível no Ótimo, não foi usada neste exemplo pois as descrições de alguns parâmetros de saída ainda não foram disponibilizados pelos seus desenvolvedores.

Finalmente, o trabalho está pronto para ser enviado ao sistema. A Figura 4.2 mostra a versão sem detalhes da página de submissão do ALGENCAN 1.0, construída para facilitar o uso do sistema por usuário experientes.

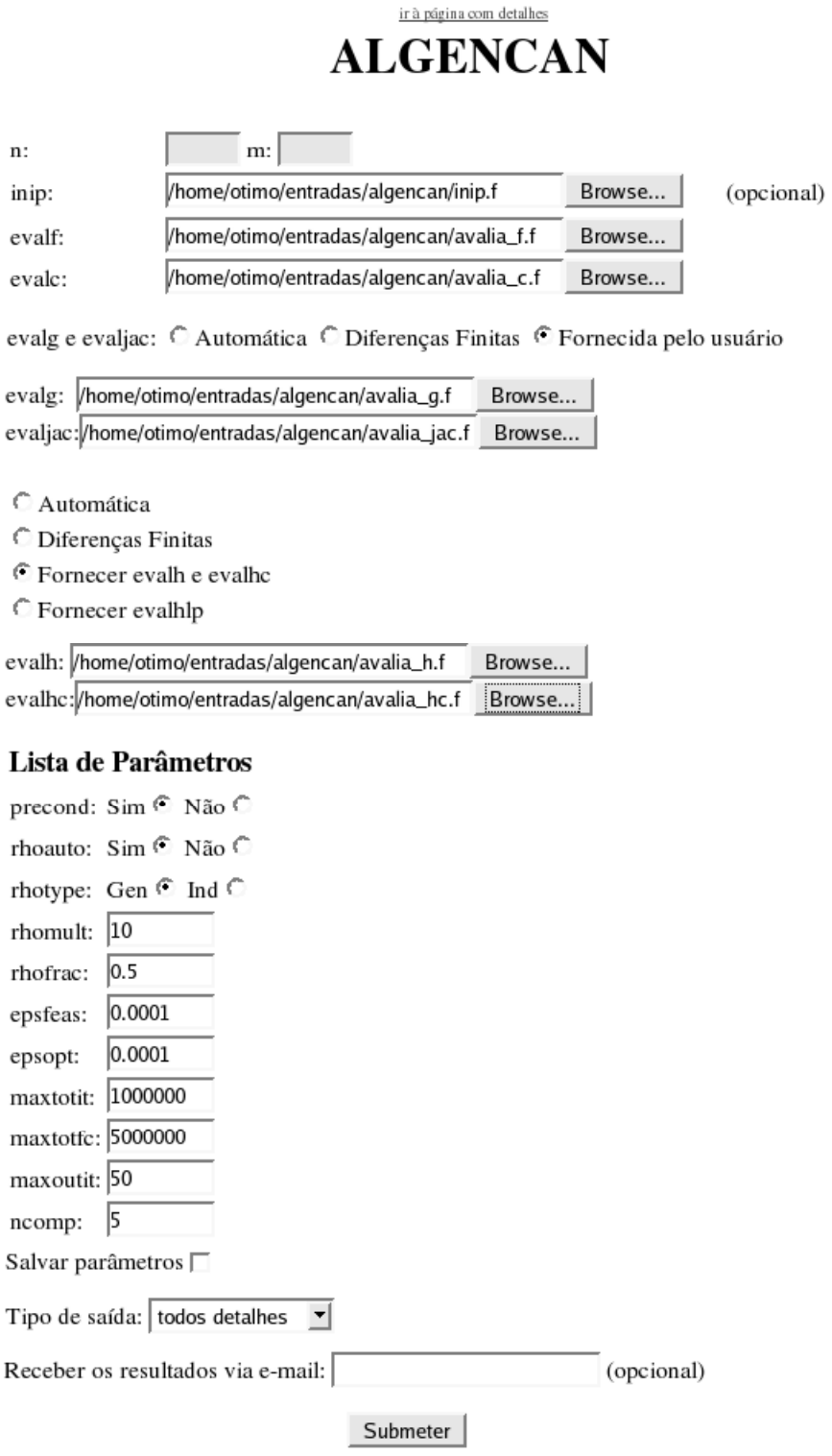



Figura 4.2: Página de submissão.

Quando o usuário clica no botão "submeter", a página da Figura 4.3 é mostrada na tela.

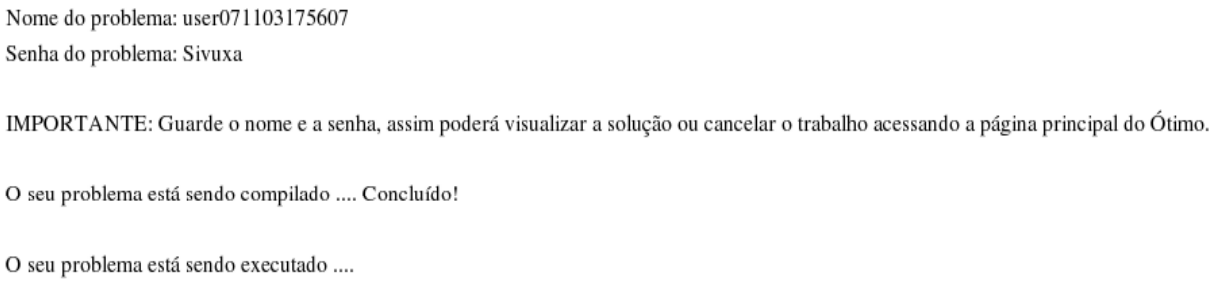
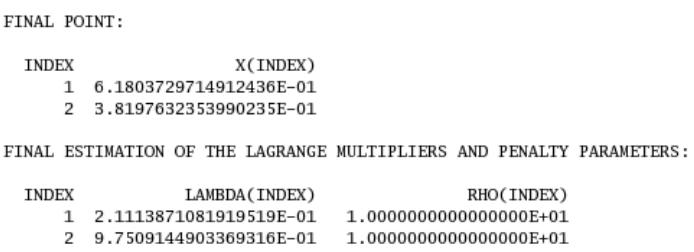



Figura 4.3: Página seguinte à submissão, mostrando o andamento do processo.

A mensagem acima indica que o problema foi compilado com sucesso e que, no momento, está sendo executado. Como o problema desse exemplo é extremamente simples, o programa de otimização resolve-o quase instantaneamente e uma página de saída, como a mostrada na Figura 4.4, é aberta:


Figura 4.4: Página com o relatório final.

O relatório de saída indica que o algoritmo encontrou um ponto que satisfazia os critérios de otimalidade (flag=0). Ao clicar em "Ver Solução", a seguinte página é exibida:

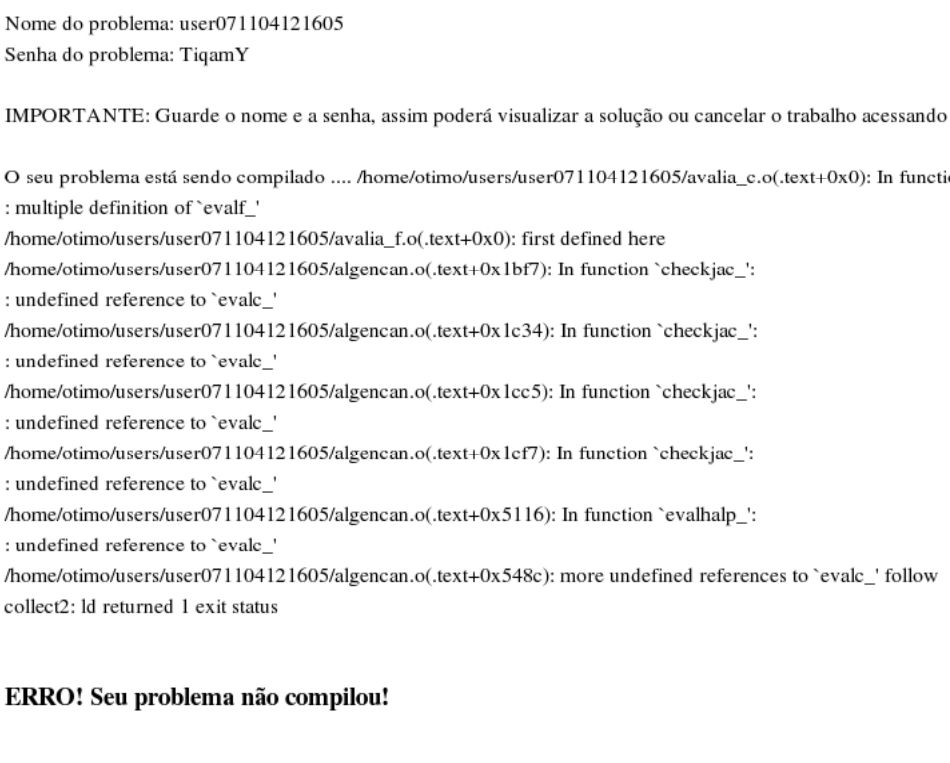


```
FINAL POINT:

  INDEX                X(INDEX)
      1  6.1803729714912436E-01
      2  3.8197632353990235E-01

FINAL ESTIMATION OF THE LAGRANGE MULTIPLIERS AND PENALTY PARAMETERS:

  INDEX           LAMBDA(INDEX)              RHO(INDEX)
      1  2.1113871081919519E-01   1.0000000000000000E+01
      2  9.7509144903369316E-01   1.0000000000000000E+01
```

Figura 4.5: Página com a solução.

A página acima mostra a solução, além dos valores finais dos multiplicadores de Lagrange e dos parâmetros de penalização.

A título de ilustração, suponha que, na página de submissão, o usuário envie um arquivo errado. Por exemplo, ele pode enviar "evalf" ao invés de "evalc". Neste caso, no sistema constariam duas funções "evalf" e nenhuma "evalc", e a seguinte página seria exibida:



Nome do problema: user071104121605
Senha do problema: TiqamY

IMPORTANTE: Guarde o nome e a senha, assim poderá visualizar a solução ou cancelar o trabalho acessando a página principal do Ótimo.

O seu problema está sendo compilado .... /home/otimo/users/user071104121605/avalia_c.o(.text+0x0): In function `evalf_':
: multiple definition of `evalf_'
/home/otimo/users/user071104121605/avalia_f.o(.text+0x0): first defined here
/home/otimo/users/user071104121605/algencan.o(.text+0x1bf7): In function `checkjac_':
: undefined reference to `evalc_'
/home/otimo/users/user071104121605/algencan.o(.text+0x1c34): In function `checkjac_':
: undefined reference to `evalc_'
/home/otimo/users/user071104121605/algencan.o(.text+0x1cc5): In function `checkjac_':
: undefined reference to `evalc_'
/home/otimo/users/user071104121605/algencan.o(.text+0x1cf7): In function `checkjac_':
: undefined reference to `evalc_'
/home/otimo/users/user071104121605/algencan.o(.text+0x5116): In function `evalhalp_':
: undefined reference to `evalc_'
/home/otimo/users/user071104121605/algencan.o(.text+0x548c): more undefined references to `evalc_' follow
collect2: ld returned 1 exit status

**ERRO! Seu problema não compilou!**

Figura 4.6: Página de saída de um problema que apresentou erro na fase de compilação.

Como era esperado, ocorreu um erro na fase de compilação. O texto reproduz a mensagem de erro gerada pelo compilador.

# Capítulo 5

# Como inserir um novo programa de otimização

Para um programa de otimização estar apto a operar no sistema, quatro arquivos devem ser criados obrigatoriamente. Definindo o termo *solver* como o nome do programa que está sendo inserido no sistema, os arquivos a serem criados são: *solver*.html (página de entrada), *solver*.cgi (entrada de dados), comp_*solver*.pl (compilação) e gerahtml_*solver*.pl (página de saída). Neste capítulo, descreveremos como criar esses arquivos.

Para o programa operar com funcionalidades adicionais (como a derivada automática, por exemplo), outros arquivos devem ser criados e alterados. Ao longo deste capítulo, tais modificações também serão descritas.

## 5.1 Página do programa: *solver.html*

Este arquivo do tipo HTML é responsável pela página do programa. Deve conter informações sobre o programa e sobre os arquivos que devem ser enviados, caixas para envio das funções e para modificação do valor dos parâmetros, além de outras informações que o administrador do sistema considerar relevante. Após serem inseridos, os dados do usuário devem ser enviados ao arquivo *solver*.cgi, que será descrito na próxima seção.

Recomenda-se o uso de funções JavaScript para validação do formulário e também para tornar a página dinâmica, como foi feito para os programas GENCAN e ALGENCAN. Essas funções em JavaScript devem sempre ser colocadas no fim do arquivo "funcoes.js" (já existente) e recomenda-se usar a nomenclatura padrão do sistema para o nome das funções. Por exemplo, se a última função em "funcoes.js" chama-se "JS15", então a próxima função terá o nome "JS16".

Para os programas GENCAN e ALGENCAN, também foram geradas páginas alternativas, sem todas as informações das funções a serem enviadas. Essas páginas são dedicadas aos usuários mais experientes, que já tenham ciência de como enviar seu problema. O nome de uma página desse tipo deve ser "*solver*_det.html" e as funções JavaScript a ela correspondentes devem ser inseridas no arquivo (já existente) "funcoes_det.js".

## 5.2 Captura dos dados: *solver.cgi*

Os arquivos do tipo "*solver*.cgi", responsáveis pela entrada de dados no sistema, já foram discutidos em detalhes na Seção 3.1. Como já mencionado, esses arquivos diferes pouco entre os diferentes programas de otimização. Assim, para criar um novo "*solver*.cgi", recomenda-se que o mesmo seja feito baseado em um arquivo já existente, como o "algencan.cgi", por exemplo. A seguir, são mostrados alguns trechos do código de "algencan.cgi" que devem ser alterados para a inclusão de um novo programa de otimização.

Como visto na Seção 3.1, na primeira parte de "*solver*.cgi" é criado o trabalho do usuário. Esse trecho é idêntico para todos os programas, pois é usada a função "cria_pasta", da biblioteca do Ótimo (o arquivo "otimo.pm"). Essa função cria a pasta do usuário no sistema e retorna o nome do trabalho.

Após a criação da pasta de trabalho, segue a seção "baixa arquivos". Inicia-se esta etapa definindo um vetor com o nome de todos os arquivos que possam ser enviados pelo usuário. Tais nomes devem coincidir com os nomes dos campos para envio desses arquivos em "solver.html". Veja como é esse vetor para o ALGENCAN[1]:

```
@nomes =("inip","avalia_f","avalia_c","avalia_g","avalia_jac","avalia_h",
"avalia_hc","avalia_hlp");
```

Antes dos arquivos serem carregados no sistema, é necessário verificar se a linguagem de programação em que foram escritos é aceita pelo programa de otimização. O código a seguir define a linguagem e faz tal verificação.

```
$file = $q->param("avalia_f");
@aux=split(/\./,$file);
if(($aux[$#aux]eq"f")or($aux[$#aux]eq"for" )or($aux[$#aux]eq"FOR")){
  $ling="Fortran";
  $extensao="f";
}
elsif ($aux[$#aux]eq"c"){
  $ling="c";
```

[1]Neste capítulo, os exemplos dados referem-se a versão 1.0 do ALGENCAN.

```
  $extensao="c";
}
elsif (($aux[$#aux]eq"cpp")or($aux[$#aux]eq"c++")or($aux[$#aux]eq"C")
or($aux[$#aux]eq"cc")or($aux[$#aux]eq"cp")or($aux[$#aux]eq"cxx")){
  $ling="c++";
  $extensao="cpp";
}
else{
  print $q->header;
  dienice("Extensão não reconhecida! Exemplos de extensões conhecidas: .f
  .c .cpp");
}
```

Neste código, @*aux* é um vetor cujo último elemento $*aux*[$#*aux*] contém a extensão de um arquivo enviado ("avalia_f"). Como o ALGENCAN pode receber arquivos escritos em Fortran, C, ou C++, o código acima verifica se o arquivo enviado tem uma extensão compatível com tais linguagens. Se o arquivo enviado tem uma extensão conhecida, verifica-se a linguagem dos outros arquivos enviados. Caso contrário, cancela-se a submissão e uma mensagem de erro é enviada.

A adaptação do código para um novo programa de otimização é direta. O nome "avalia_f" deve ser substituído por um elemento do vetor @*nomes*, definido para o novo programa, que seja de envio mandatório. Já nas cláusulas condicionais, deve-se verificar uma possível extensão aceita pelo programa.

No ALGENCAN, todos os arquivos enviados devem ter sido escritos na mesma linguagem. O próximo código faz essa verificação.

```
foreach $arq(@nomes){
  $file = $q->param("$arq");
  if($file ne ""){
    @aux=split(/\./,$file);
    if($ling eq "Fortran"){
      if (($aux[$#aux]ne"f")and($aux[$#aux]ne"for" )and
      ($aux[$#aux]ne"FOR")){
        print $q->header;
        dienice("Extensões diferentes!");
      }
    }
    elsif($ling eq "c"){
      if ($aux[$#aux]ne"c"){
        print $q->header;
```

```
        dienice("Extensões diferentes!");
      }
    }
    elsif($ling eq "c++"){
      if(($aux[$#aux]ne"cpp")and($aux[$#aux]ne"c++")and
      ($aux[$#aux]ne"C")and($aux[$#aux]ne"cc")
      and($aux[$#aux]ne"cp")and($aux[$#aux]ne"cxx")){
        dienice("Extensões diferentes!");
      }
    }
    else{
      print $q->header;
      dienice("Extensão não reconhecida! Exemplos de extensões
      conhecidas: .f .c .cpp");
    }
  }
}
```

Neste código, para cada elemento do vetor @*nomes*, é feita a verificação se a extensão do arquivo está de acordo com a linguagem contida na variável $*ling*, atribuída no código anterior.

De maneira análoga ao código anterior, a adaptação aqui para um novo programa resume-se em substituir, nas cláusulas condicionais, as extensões aceitas pelo novo programa.

Depois de feitas as verificações mencionadas até aqui, os arquivos do usuário já podem ser salvos em disco no servidor do Ótimo. Como esse código é o mesmo para todos os programas, não será exibido aqui. Caso o leitor tenha interesse em saber como é feito o carregamento dos arquivos, ou outros detalhes, pode baixar todo o sistema no endereço do Ótimo.

Seguindo a sequência da Figura 3.1, o próximo passo é a geração do arquivo com o valor dos parâmetros. Para o ALGENCAN, o arquivo de parâmetros, em Fortran, tem o formato abaixo:

```
      subroutine param(rhoauto,rhotype,rhomult,rhofrac,m,rho,gtype,
     +hptype,intype,precond,checkder,epsfeas,epsopt,maxoutit,maxtotit,
     +maxtotfc,iprint,ncomp)

      character * 6 precond
      logical checkder,rhoauto
```

```
      integer gtype,hptype,iprint,intype,m,maxoutit,maxtotfc,maxtotit,
     +ncomp,rhotype
      double precision epsfeas,epsopt,rhofrac,rhomult, rho(m)

      gtype    = 0
      hptype   = 6
      intype   = 0
      precond  = ''QNCGNA''
      rhoauto  = .true.
      rhotype  = 1
      rhomult  = 1.0d+01
      rhofrac  = 0.5d0
      checkder = .false.
      epsfeas  = 1.0d-04
      epsopt   = 1.0d-04
      maxoutit = 50
      maxtotit = 1000000
      maxtotfc = 5000000
      iprint   = 1
      ncomp    = 4

      end
```

Na página de submissão, o usuário pode escolher o valor desses parâmetros via botão (e.g. tipo de derivada) ou via campo de texto (e.g. número máximo de iterações). Assim, para o ALGENCAN, o sistema deve converter estes dados para o formato indicado pelo código acima.

Para fazer essa conversão, primeiramente, é definido um vetor com o nome de todos os parâmetros do programa (devem ser usados os mesmos nomes definidos em *solver*.html). Para o ALGENCAN, o vetor é:

```
@parame=("gtype","hptype","intype","precond","rhoauto","rhotype",
"rhomult","rhofrac","checkder","epsfeas","epsopt","maxoutit","maxtotit",
"maxtotfc","iprint","ncomp");
```

Em seguida, é criada a primeira parte do arquivo de parâmetros, contendo o nome da rotina e declaração das variáveis. Segue um exemplo:

```
if($ling eq "Fortran"){
  open(PARAM,">$user_home/param.f");
  print PARAM "      subroutine param(rhoauto,rhotype,rhomult,rhofrac,
```

```
                    +m,rho,gtype,hptype,intype,precond,checkder,epsfeas,
                    +epsopt,maxoutit,maxtotit,maxtotfc,iprint,ncomp)\n";
  print PARAM "      implicit none\n";
  print PARAM "      logical checkder,rhoauto\n";
  print PARAM "      character * 6 precond\n";
  print PARAM "      integer gtype,hptype,iprint,intype,m,maxoutit,
                    +maxtotfc,maxtotit,ncomp,rhotype,i\n";
  print PARAM "      double precision epsfeas,epsopt,rhofrac,rhomult\n";
  print PARAM "      double precision rho(m)\n";
}
```

Na primeira linha do código acima, é definido que o arquivo de parâmetros será criado em Fortran, supondo que o usuário enviou os arquivos nessa linguagem. O código em C é análogo ao Fortran, e não será exibido aqui.

Na segunda linha, o arquivo "param.f" é aberto para escrita e, nas demais linhas, são declaradas as variáveis e a subrotina. Para completar o arquivo, resta atribuir valor às variáveis. É justamente essa a função do código a seguir.

```
foreach $i(@parame){
  if($i eq "gtype"){
    $derive1 = $q->param("deriv1");
    if($derive1 eq "2"){
      $fil="1";
    }
    else{
      $fil="0";
    }
  }
  .
  .
  elsif($i eq "rhotype"){
    $rhotype = $q->param($i);
    if($rhotype eq "gen"){
      $fil = "1";
    }
    elsif($rhotype eq "ind"){
      $fil = "2";
    }
  }
  .
```

```
  .
  else{
    $fil = $q->param($i);
    $dados_cookie.="$i:$fil,";
  }
  # Escreve "nome do parâmetro"="valor do parâmetro" em Fortran
  print PARAM "      $i=$fil\n";
}
```

No código anterior, há um laço que percorre o vetor de parâmetros @*parame* e que, a cada iteração, atribui ao parâmetro $\$i$ o valor escolhido pelo usuário $\$fil$. No entanto, quando o campo a ser preenchido não requerer valores numéricos (caso dos parâmetros "gtype" e "rhotype" do código acima), pode ser necessário um tratamento específico.

Por exemplo, a primeira condição do laço refere-se ao parâmetro "gtype", que define o tipo da primeira derivada. Se "gtype" for zero, o ALGENCAN supõe que o usuário forneceu o código com a derivada. Caso "gtype" seja um, o ALGENCAN calcula a derivada por diferenças finitas. No entanto, como já foi visto, na visão do usuário existem três opções para o cálculo da primeira derivada (automática, diferenças finitas ou fornecê-la) e, para cada uma, está associado um valor (1, 2 e 3, respectivamente) para o campo "deriv1" em *solver*.html. Dessa forma, a primeira cláusula condicional do código acima faz a conversão de "deriv1" para "gtype".

Já quando o parâmetro na página do programa é numérico, basta atribuir o valor passado pelo usuário à variável (primeira linha da cláusula "else" do código acima).

Repare ainda que, nesse código, é criada a variável $\$dados\_cookie$, que concatena as variáveis numéricas (separadas por virgula) para posterior criação do *cookie*, usado para salvar os dados de uma submissão do usuário.

A etapa seguinte no fluxo de armazenamento dos dados é a criação do arquivo "dados.txt". Para tal, só falta a definição de uma senha para o usuário. A senha é obtida com a função "random_password" (veja [10]), da biblioteca do Ótimo. Assim, para o ALGENCAN, o código que define "dados.txt" será:

```
open(DADOS,">$user_home/dados.txt");
# Dados padrão (obrigatório, inclusive a ordem) p/ todos os solvers
print DADOS "$user\n";                  # [0]: nome do trabalho
print DADOS "$passwd\n";                # [1]: senha do trabalho
print DADOS "$solver\n";                # [2]: solver escolhido
print DADOS "$email\n";                 # [3]: email do usuário
print DADOS "$ling\n";                  # [4]: linguagem escolhida
```

```
# Dados específicos do ALGENCAN
print DADOS "$derive1\n";              # [5]: tipo da primeira derivada
print DADOS "$derive2\n";              # [6]: tipo da segunda derivada
print DADOS "$n\n";                    # [7]: dimensão
print DADOS "$m\n";                    # [8]: número de restrições
close(DADOS);
```

Na inclusão de um novo programa, os dados padrão são obrigatórios e devem seguir esta ordem. Já os dados específicos são opcionais e dependem do programa.

Seguindo o fluxo, caso o usuário tenha fornecido seu endereço eletrônico, é enviado à ele uma mensagem indicando o sucesso no envio do trabalho, com alguns *links* para posterior acesso (através da função "envia_email", da biblioteca do Ótimo). Porém, antes do envio da mensagem, é feita uma validação no endereço fornecido (função "verifica_email").

Finalmente, o problema é inserido na fila de compilação, encerrando a fase de armazenamento de dados.

## 5.3 Compilação e geração do arquivo executável: *comp_solver.pl*

Como já foi visto na Seção 3.2, o *script* "comp_solver.pl", chamado por "compila.pl", deve, ao fim de sua execução, gerar na pasta do usuário um arquivo executável com o nome "executar", pronto para ser rodado quando chegar a sua vez na fila de execução. Assim sendo, a estrutura que será descrita nesta seção servirá mais como uma sugestão na inclusão de novos otimizadores, pois a única obrigatoriedade é a criação do arquivo "executar".

Na primeira etapa, são lidos os dados do usuário. No ALGENCAN, os dados carregados são: o programa chamado, a linguagem escolhida, o tipo da derivada de primeira ordem, o tipo da derivada de segunda ordem, a dimensão do problema e o número de restrições. Em seguida, chama-se a função "extensao" (da biblioteca do Ótimo), cuja entrada é a linguagem escolhida e cuja saída são as variáveis $\$ext$ e $\$comp$, que indicam, respectivamente, a extensão dos arquivos gerados na pasta do usuário e o compilador que será utilizado.

A etapa seguinte refere-se à atribuição de valores iniciais a alguns parâmetros. Se, para um novo programa de otimização, o administrador achar interessante definir alguns valores padrão para tais parâmetros, analogamente ao que foi feito no GENCAN e ALGENCAN, recomenda-se que crie uma função auxiliar com a atribuição de todos os

valores padrão e, no fim dessa função, faça uma chamada à função enviada pelo usuário. Também é importante que o padrão definido seja o mais geral possível. Por exemplo, se um parâmetro é responsável por indicar se uma restrição é linear ou não-linear, é mais indicado definir como padrão que todas restrições sejam não-lineares.

Ao inserir um novo programa de otimização, é provável que, antes da compilação, seja necessário alterar algumas linhas do código original desse programa. Essas alterações devem ser feitas no *script* "config_solver.pl", que será chamado por "comp_solver.pl". Por exemplo, no ALGENCAN, esse *script* é usado para mudar a pasta de saída do otimizador para a pasta criada para o usuário pelo sistema, entre outras coisas.

Para a geração das derivadas automáticas, o arquivo "gerader_solver.pl" deve ser criado. Ao final de sua execução, devem constar, na pasta do usuário, os arquivos gerados pelo Adol-C que serão usados posteriormente para a avaliação das derivadas. Se for tomado como referência o arquivo do ALGENCAN, "gerader_algencan.pl", poucas alterações terão que ser realizadas. Na primeira delas, de caráter obrigatório, deve-se substituir a linha em que a função "evalf" é chamada em "aux_cpp.cpp", mostrado na pág. 48. A nova linha deve conter uma chamada para função objetivo do programa que está sendo inserido, com seus respectivos parâmetros. A função "main_cpp.cpp", que faz uma chamada a "inip" para trazer a dimensão e o ponto inicial do problema, também deve ter essa linha alterada, incluindo o nome da nova função que define tais valores iniciais. Por fim, deve-se mudar o código de "gerader_solver.pl" para que compile e gere as derivadas dessas novas funções, como mostrado na pág. 49.

Para o cálculo das derivadas automáticas, devem ser criados arquivos análogos a "avalia_gder.cpp", exibido na pág. 50. Apenas deve-se substituir o nome e os parâmetros da função para aqueles relacionados ao novo programa de otimização.

No que se refere à compilação dos arquivos do novo programa, há três situações distintas, conforme descrito na Seção 3.2.5. Os tipos de derivada escolhida implicarão na compilação de arquivos diferentes. Assim, como cada programa exige o envio de arquivos específicos, cada um terá seu próprio código de compilação. Para ilustrar, segue o trecho do código de compilação do ALGENCAN para o usuário que escolheu, na página de submissão, a derivada de primeira ordem automática ($\$derive1 = 1$) e a derivada de segunda ordem por diferenças finitas ($\$derive2 = 2$):

```
if($derive1 eq "1" and $derive2 eq "2"){
  # Gera as derivadas automáticas
  perl $otimo_home/bin/gerader_algencan.pl $user_home;

  # Compila "avalia_g" de maneira que esta leia as derivadas geradas
```

```
  # pelo Adol-C
  g++ -I$otimo_home/adolc_base/include -c
  $otimo_home/solvers/algencan/autodif/avalia_der/$ling/avalia_gder.cpp
  -o $user_home/avalia_g.o;

  # Compila "avalia_jac" de maneira que esta leia as derivadas geradas
  # pelo Adol-C
  g++ -I$otimo_home/adolc_base/include -c
  $otimo_home/solvers/algencan/autodif/avalia_der/$ling/avalia_jacder.cpp
  -o $user_home/avalia_jac.o;

  # Compila arquivos vazios (mas necessários, pois são chamados pelo solver)
  $comp -c $otimo_home/solvers/algencan/avalia_vazio/avalia_h_vazio.$ext
  -o $user_home/avalia_h.o;
  $comp -c $otimo_home/solvers/algencan/avalia_vazio/avalia_hc_vazio.$ext
  -o $user_home/avalia_hc.o;
  $comp -c $otimo_home/solvers/algencan/avalia_vazio/avalia_hlp_vazio.$ext
  -o $user_home/avalia_hlp.o;
  }
}

# Compila "avalia_f" e "avalia_c" enviados pelo usuário
$comp  -c $user_home/avalia_f.$ext -o $user_home/avalia_f.o;
$comp  -c $user_home/avalia_c.$ext -o $user_home/avalia_c.o;

# Compila os arquivos do ALGENCAN
$comp -I$otimo_home/solvers/algencan/c -c $user_home/algencanma.$ext
-o $user_home/algencanma.o;
g77 -c -xf77-cpp-input $user_home/algencan.f -o $user_home/algencan.o;
```

Nesse código, primeiramente é chamado o *script* para cálculo das derivadas automáticas ("gerader_algencan.pl"). Em seguida, são compiladas as funções para avaliação do gradiente ("avalia_gder.cpp") e do jacobiano ("avalia_jacder.cpp"), que foram calculadas via Adol-C. Daí, são compiladas as funções que não serão usadas pelo otimizador mas que devem estar presentes ("avalia_h_vazio", "avalia_hc_vazio", "avalia_hlp_vazio". Então, são calculadas as funções enviadas pelo usuário ("avalia_f" e "avalia_c"), Finalmente, são compilados os arquivos do programa de otimização.

Para finalizar o *script* "comp_solver.pl", basta fazer o *link* das funções compiladas para gerar o arquivo executável. Para ilustrar, segue esse comando no ALGENCAN:

```
# Faz o link entre as funções compiladas e gera o executável "executar"
g77 -I$otimo_home/adolc_base/include -lstdc++ $user_home/inip.o
$user_home/avalia_f.o $user_home/avalia_c.o $user_home/avalia_g.o
$user_home/avalia_jac.o $user_home/avalia_h.o $user_home/avalia_hc.o
$user_home/avalia_hlp.o $user_home/param.o $user_home/algencanma.o
$user_home/algencan.o $wrapper $otimo_home/adolc_base/lib/libadolc.a
-o $user_home/executar;
```

## 5.4 Página de saída: *gerahtml_solver.pl*

A rotina "gerahtml_solver.pl" é reponsável pela criação da página de saída, podendo conter a solução ou alguma mensagem de erro, conforme já discutido na Seção 3.6.

A saída de erro é padrão para todos os programas de otimização, sendo simplesmente a mensagem de erro gerada pelo compilador ou outro programa que tenha detectado o erro. Assim, as alterações que devem ser feitas em "gerahtml_solver.pl" para inclusão de um novo programa são referentes somente à saída contendo a solução (ou seja, quando o *script* é chamado com o parâmetro "ok"). Como exemplo, segue o trecho do código do *script* "gerahtml_algencan.pl":

```
# Se o problema não pôde ser resolvido, saida.html conterá a mensagem
# com o erro encontrado
if($tipo_saida ne "ok"){
  .
  .
}
# O problema foi resolvido
else{
  ######################################################################
  # IMPORTANTE: TRECHO A SER MODIFICADO NA INCLUSÃO DE UM NOVO SOLVER #
  # CRIAR NOVO ARQUIVO COM O NOME GERAHTML_NOMEDOSOLVER               #
  ######################################################################

  # Define se haverá botão adicional para acesso à informações detalhadas
  # Se o código abaixo retornar $detail=0 significa que não haverá tal botão
  .
  .
  # Escreve em saida.html as informações do solver
  print HTML "Nome do usuário: $user <br>";
  print HTML "Senha: $senha <br>";
  print HTML "<h2>Relatório do ALGENCAN:</h2>";
```

```
  print HTML "<pre>\n";
  print HTML "<table>";
  print HTML "<tr><td>Dimensão do problema</td><td>=</td><td>$n</td></tr>";
  print HTML "<tr><td>Número de restrições</td><td>=</td><td>$m</td></tr>";
  .
  .
  print HTML "</table><br><br><br>";
  ######################################################################
  #               FIM DO TRECHO A SER MODIFICADO                       #
  ######################################################################
```

Veja na Figura 4.4 como fica a página de saída gerada por este código. Ainda nessa figura, note que há dois botões no final da página. O primeiro deles, "Ver saída detalhada", é opcional, e só é exibido quando $\$detail > 0$ no código acima. O *script* executado quando este botão é acionado é "detail.cgi", cujo trecho a ser alterado é mostrado abaixo:

```
.
.
if($solver eq "GENCAN"){
  open(SAIDA,"$user_home/algencan.out");
}
elsif($solver eq "ALGENCAN"){
  open(SAIDA,"$user_home/algencan.out");
}
.
.
```

Na inserção de um novo programa de otimização, basta criar um nova cláusula condicional no código acima com o nome do novo programa e do arquivo com a saída detalhada.

A alteração em "soluc.cgi", responsável pelo segundo botão, "Ver solução", é idêntica à feita para "detail.cgi".

# Conclusão

Neste trabalho, foi apresentado um sistema que integra, via internet, diferentes programas de otimização. Para cada programa, há uma página que descreve como enviar um problema e como configurar suas opções. Para os novos usuários, é possível fazer uma submissão simplificada, ao escolher derivadas automáticas e usar valores padrão dos parâmetros do programa. Para os usuários experientes, há a possibilidade de se fazer inúmeros testes apenas trocando as opções na página do programa.

Uma componente importante integrada ao sistema é o pacote Adol-C, que pode ser usado para calcular as derivadas das funções enviadas pelo usuário. Como, em geral, os programas de otimização necessitam das derivadas das funções que vai otimizar, sugere-se que o Adol-C seja utilizado nos próximos programas a serem inseridos.

O sistema ainda possui o esquema das duas filas (de compilação e execução, que correm em paralelo), a possibilidade de cancelamento de um trabalho, o envio dos resultados por correio eletrônico, entre outras inúmeras funcionalidades já discutidas neste texto.

É importante salientar que este trabalho deve servir como ponto de partida para um amplo sistema de otimização. Para tal, é fundamental a inclusão de novos programas de otimização no sistema. Como foi visto, a dificuldade em inserir novos programas está relacionada ao número de opções que o desenvolvedor do programa deseja deixar disponíveis ao usuário. Assim, espera-se que, ao término da leitura deste texto, um leitor com conhecimento básico da linguagem Perl possa inserir um novo programa com opções análogas às do GENCAN e do ALGENCAN.

Entretanto, o objetivo deste texto não é simplesmente servir como um manual de referência para o Ótimo. Foram discutidas as dificuldades envolvidas ao desenvolver um sistema desse tipo, que integra programas muito diferentes com o propósito de resolver o problema do usuário. O texto pode ainda ser útil para usuários que desejem utilizar separadamente programas que estão no sistema, como o Adol-C e o ALGENCAN, por conter explicações sucintas e exemplos.

Como um trabalho futuro, seria importante inserir um módulo no sistema que calcu-

lasse as principais estatísticas dos trabalhos enviados, e as enviassem periodicamente aos desenvolvedores dos programas de otimização. Entretanto, os desenvolvedores não deverão ter acesso aos detalhes dos trabalhos submetidos, a fim de garantir a confidencialidade das informações enviadas.

Outro módulo que poderia ser programado no sistema refere-se à prioridade na execução dos problemas. Já foi mencionado que, na compilação, é obtida a dimensão e o número de restrições do problema enviado. Assim, o sistema poderia varrer a fila de execução e dar prioridade para os problemas menores, que, provavelmente, serão resolvidos mais rapidamente. Dessa maneira, evitar-se-ia que um problema grande congestionasse a fila de execução.

Por fim, poderia ser criada uma página *web* para a administração do sistema, com acesso restrito, de modo que as principais configurações do sistema pudessem ser visualizadas e alteradas.

# Apêndice A
# Estrutura de diretórios e arquivos

Nesta seção, são enumerados todos os diretórios e arquivos que fazem parte do sistema, acompanhados de uma breve descrição. Um nome com uma barra no final representa um diretório.

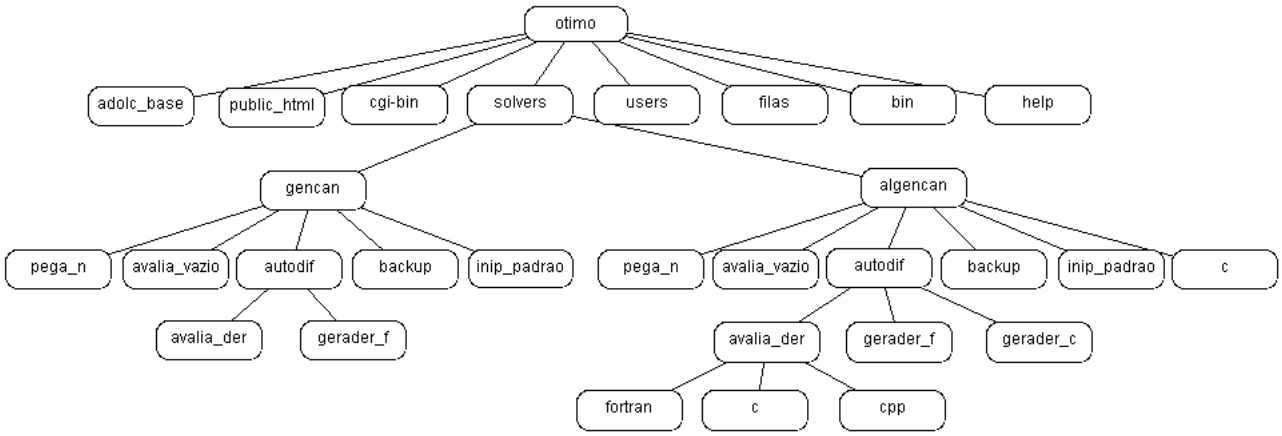



Figura 1: Estrutura de diretórios

- **adolc_base/ :** Contém o programa Adol-C instalado. Seus arquivos de configuração, biblioteca e manual de uso.
- **backup/ :** Contém backups do sistema em diferentes datas.
- **bin/ :** Contém os arquivos executáveis do sistema.
  - **compila.pl :** Responsável pela fila de compilação, é chamado enquanto houver trabalho nessa fila.
  - **comp_algencan.pl :** Compila os arquivos do programa ALGENCAN.
  - **comp_gencan.pl :** Compila os arquivos do programa GENCAN.

  - **cron.pl :** Restabelece o sistema caso o servidor tenha saído do ar.
  - **execut.pl :** Responsável pela fila de execução, é chamado enquanto houver trabalho nessa fila.
  - **gerahtml_gencan.pl :** Gera a página de saída do GENCAN.
  - **gerahtml_algencan.pl :** Gera a página de saída do ALGENCAN.
  - **roda :** Arquivo bash que roda o programa de otimização.
  - **roda_deriv :** Arquivo bash que roda o executável gerado pelo Adol-c para o GENCAN.
  - **roda_deriv_evalf :** Arquivo bash que roda o executável gerado pelo Adol-c para o ALGENCAN.
  - **roda_deriv_evalc :** Arquivo bash que roda o executável gerado pelo Adol-c para o ALGENCAN.

- **cgi-bin/ :** Contém os arquivos CGI do sistema.
  - **algencan.cgi :** Recebe os arquivos do usuário para o uso do ALGENCAN.
  - **cancela.cgi :** Cancela um trabalho submetido.
  - **cgi-lib.pl :** Biblioteca CGI.
  - **coment.cgi :** Recebe os comentários enviados ao sistema.
  - **detail.cgi :** Exibe saída detalhada dos programas de otimização.
  - **fila.cgi :** Exibe filas de compilação e execução.
  - **gencan.cgi :** Recebe os arquivos do usuário para o uso do GENCAN.
  - **result.cgi :** Exibe página de saída.
  - **soluc.cgi :** Exibe o vetor contendo a solução do problema.

- **filas/ :** Contém as filas de compilação e exucução.
  - **filac.txt :** Trabalhos na fila de compilação.
  - **filae.txt :** Trabalhos na fila de execução.

- **help/ :** Contém arquivos de ajuda e sugestões.
  - **bugs.txt :** Relatório com os bugs conhecidos.
  - **users.txt :** Sugestões feitas pelos usuários na página do sistema.

- **public_html/ :** Contém páginas em HTML e funções em JavaScript.
  - **algencan.html :** Programa ALGENCAN.

  - **algencan_det.html :** Programa ALGENCAN sem detalhes.
  - **contato.html :** Envio de sugestões.
  - **cancela.html :** Cancelamento de trabalhos.
  - **gencan.html :** Programa GENCAN.
  - **gencan_det.html :** Programa GENCAN sem detalhes.
  - **tipodif.html :** Informações sobre produto Hessiana-vetor usando diferenças finitas (ALGENCAN).
  - **index.html :** Página inicial do sistema.
  - **logos/ :** Contém as figuras usadas nas páginas.
  - **funcoes.js :** Funções em JavaScript.
  - **funcoes_det.js :** Funções em JavaScript para páginas sem detalhes.
  - **result.html :** Ver resultado de trabalho.

- **solvers/ :** Contém os programas de otimização do sistema.
  - **algencan/ :** Programa ALGENCAN.
  - **../algencan_aux.f :** Arquivo do programa sem a dimensão máxima dos vetores.
  - **../algencanma_aux.f :** Arquivo do programa sem a dimensão máxima dos vetores.
  - **../config_algencan.pl :** Prepara o programa para ser usado no sistema.
  - **algencan/autodif/ :** Contém arquivos para diferenciação automática.
  - **../converte_c.pl :** Converte evalc para diferenciação.
  - **../converte_f.pl :** Converte evalf para diferenciação.
  - **algencan/autodif/avalia_der/ :** Contém funções que avaliam as derivadas geradas pelo Adol-c.
  - **algencan/autodif/avalia_der/c/ :** Avalia derivadas para funções enviadas em C.
  - **../avalia_gder.cpp :** Gradiente de evalf
  - **../avalia_hlpder.cpp :** Hessiana do Lagrangiano vezes um vetor
  - **../avalia_jacder.cpp :** Gradiente de evalc
  - **algencan/autodif/avalia_der/cpp/ :** Avalia derivadas para funções enviadas em C++. Os arquivos deste diretório têm o mesmo nome dos arquivos listados no diretório anterior, assim serão omitidos.

- **algencan/autodif/avalia_der/Fortran/ :** Avalia derivadas para funções enviadas em Fortran. Os arquivos deste diretório têm o mesmo nome dos arquivos listados no diretório anterior, assim serão omitidos.
- **algencan/autodif/gerader_evalc/ :** Contém funções que geram as derivadas de evalc.
- **../aux_evalc_c.cpp :** Função auxiliar para evalc em C.
- **../aux_evalc_cpp.cpp :** Função auxiliar para evalc em C++.
- **../aux_evalc_f.cpp :** Função auxiliar para evalc em Fortran.
- **../main_evalc_c.cpp :** Chama função auxiliar para inip em C.
- **../main_evalc_cpp.cpp :** Chama função auxiliar para inip em C++.
- **../main_evalc_f.cpp :** Chama função auxiliar para inip em Fortran.
- **algencan/autodif/gerader_evalf/ :** Contém funções que geram as derivadas de evalf. Os arquivos deste diretório têm o mesmo nome dos arquivos listados no diretório anterior, assim serão omitidos.
- **algencan/avalia_vazio/ :** Contém funções vazias do programa.
- **../avalia_g_vazio.c :** evalg em C.
- **../avalia_g_vazio.f :** evalg em Fortran.
- **../avalia_hc_vazio.c :** evalhc em C.
- **../avalia_hc_vazio.f :** evalhc em Fortran.
- **../avalia_hlp_vazio.c :** evalhlp em C.
- **../avalia_hlp_vazio.f :** evalhlp em Fortran.
- **../avalia_h_vazio.c :** evalh em C.
- **../avalia_h_vazio.f :** evalh em Fortran.
- **../avalia_jac_vazio.c :** evaljac em C.
- **../avalia_jac_vazio.f :** evaljac em Fortran.
- **algencan/backup/ :** Contém a instalação e arquivos originais do programa.
- **algencan/c/ :** Contém arquivos auxiliares para a linguagem C.
- **../algencanma.c :** Interface algencan/C.
- **../algencanma.h :** Biblioteca para algencanma.
- **../cFortran.h :** Biblioteca para a interface.
- **../cwrapper.h :** Biblioteca para a interface.
- **algencan/inip_padrao/ :** Contém funções inip com valores padrão.

- **../inip_padrao.c :** Inip em C.
- **../inip_padrao.cpp :** Inip em C++. (UNIR C com C++)
- **../inip_padrao.f :** Inip em Fortran.
- **algencan/pega_n/ :** Contém funções que capturam a dimensão de inip.
- **../pega_n.c :** Para inip em C.
- **../pega_n.cpp :** Para inip em C++. (UNIR C com C++)
- **../pega_n.f :** Para inip em Fortran.
- **gencan/ :** Programa GENCAN.
- **../algencan_aux.f :** Arquivo do programa sem a dimensão máxima dos vetores.
- **../algencanma_aux.f :** Arquivo do programa sem a dimensão máxima dos vetores.
- **../config_gencan.pl :** Prepara o programa para ser usado no sistema.
- **gencan/autodif/ :** Contém arquivos para diferenciação automática.
- **../converte_f.pl :** Converte evalf para diferenciação.
- **algencan/autodif/avalia_der/ :** Contém funções que avaliam as derivadas geradas pelo Adol-C.
- **../avalia_gder.cpp :** Gradiente de evalf
- **../avalia_hlpder.cpp :** Hessiana do Lagrangiano vezes um vetor
- **../avalia_jacder.cpp :** Gradiente de evalc
- **gencan/autodif/gerader/ :** Contém funções que geram as derivadas de evalf.
- **../aux_c.cpp :** Função auxiliar para evalf em C.
- **../aux_cpp.cpp :** Função auxiliar para evalf em C++.
- **../aux_f.cpp :** Função auxiliar para evalf em Fortran.
- **../main_c.cpp :** Chama função auxiliar para inip em C.
- **../main_cpp.cpp :** Chama função auxiliar para inip em C++.
- **../main_f.cpp :** Chama função auxiliar para inip em Fortran. (Posso fazer uma só, mas perco X0)
- **gencan/avalia_vazio/ :** Contém funções vazias do programa.
- **../avalia_g_vazio.c :** evalg em C.
- **../avalia_g_vazio.f :** evalg em Fortran.

  - **../avalia_hc_vazio.c :** evalhc em C.
  - **../avalia_hc_vazio.f :** evalhc em Fortran.
  - **../avalia_hlp_vazio.c :** evalhlp em C.
  - **../avalia_hlp_vazio.f :** evalhlp em Fortran.
  - **../avalia_h_vazio.c :** evalh em C.
  - **../avalia_h_vazio.f :** evalh em Fortran.
  - **../avalia_jac_vazio.c :** evaljac em C.
  - **../avalia_jac_vazio.f :** evaljac em Fortran.
  - **gencan/backup/ :** Contém a instalação e arquivos originais do programa.
  - **gencan/inip_padrao/ :** Contém funções inip com valores padrão.
  - **../inip_padrao.c :** Inip em C.
  - **../inip_padrao.cpp :** Inip em C++. (UNIR C com C++: Aqui há diferença!)
  - **../inip_padrao.f :** Inip em Fortran.
  - **gencan/pega_n/ :** Contém funções que capturam a dimensão de inip.
  - **../pega_n.c :** Para inip em C.
  - **../pega_n.cpp :** Para inip em C++. (UNIR C com C++: Aqui há diferença!)
  - **../pega_n.f :** Para inip em Fortran.

- **users/ :** Contém os trabalhos submetidos pelos usuários.

# Apêndice B
# Funções do Adol-C usadas no sistema

Cálculo de g = $\nabla F(x)$ :

```
int gradient(tag,n,x,g)
short int tag;                    // tape de identificação
int n;                            // número de variáveis independentes
double x[n];                      // vetor independente x
double g[n];                      // vetor resultante
```

Cálculo de z = $\nabla^2 F(x)$.v :

```
int hess_vec(tag,n,x,v,z)
short int tag;                    // tape de identificação
int n;                            // número de variáveis independentes
double x[n];                      // vetor independente x
double v[n];                      // vetor tangente v
double z[n];                      // vetor resultante
```

Cálculo de h = $u^T.\nabla^2 F(x)$.v :

```
int lagra_hess_vec(tag,m,n,x,v,u,h)
short int tag;                    // tape de identificação
int m;                            // número de variáveis dependentes
int n;                            // número de variáveis independentes
double x[n];                      // vetor independente x
double v[n];                      // vetor tangente v
double u[m];                      // vetor de multiplicadores u
double h[n];                      // vetor resultante
```

Cálculo de $\nabla^2 F(x)$ esparsa:

```
int sparse_hess(tag,n,repeat,x,&nnz,&rind,&cind,&values)
short int tag;                    // tape de identificação
```

```
int n;                              // número de variáveis independentes
int repeat;                         // guardar padrão de esparsidade
double x[n];                        // vetor independente x
int nnz;                            // número de elementos não nulos
unsigned int rind[nnz];             // índice da linha
unsigned int cind[nnz];             // índice da coluna
double values[nnz];                 // valor dos elementos não nulos
```

Cálculo de $F''(x)$ esparsa:

```
int sparse_jac(tag,m,n,repeat,x,&nnz,&rind,&cind,&values)
short int tag;                      // tape de identificação
int m;                              // número de variáveis dependentes
int n;                              // número de variáveis independentes
int repeat;                         // guardar padrão de esparsidade
double x[n];                        // vetor independente x
int nnz;                            // número de elementos não nulos
unsigned int rind[nnz];             // índice da linha
unsigned int cind[nnz];             // índice da coluna
double values[nnz];                 // valor dos elementos não nulos
```

# Referências Bibliográficas


[1] R. Andreani; E. G. Birgin; J. M. Martínez; M. L. Schuverdt. On Augmented Lagrangian methods with general lower-level constraints. *SIAM Journal on Optimization* **18**, p. 1286-1309, 2007.

[2] R. Andreani; E. G. Birgin; J. M. Martínez; M. L. Schuverdt. Augmented Lagrangian methods under the constant positive linear dependence constraint qualification. *Mathematical Programming* **111**, p. 5-32, 2008.

[3] E. G. Birgin & J. M. Martínez. Large-scale active-set box-constrained optimization method with spectral projected gradients. *Computational Optimization and Applications* **23**, p. 101–125, 2002.

[4] E. G. Birgin e J. M. Martínez. Structured Minimal-Memory inexact quasi-Newton method and secant preconditioners for Augmented Lagrangian Optimization. *Computational Optimization and Applications* **39**, pp. 1–16 (2008).

[5] J. Czyzyk; M. Mesnier; J. Moré. The NEOS Server. *IEEE J. Computational Science and Engineering* **5**, p. 68–75, 1998.

[6] E. D. Dolan. NEOS server 4.0 administrative guide. Technical memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, EUA, 2002.

[7] E. D. Dolan; R. Fourer; J. J. Moré; T. S. Munson. The NEOS Server: version 4 and beyond. Preprint ANL/MCS-P947-0202, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, EUA, 2002.

[8] J. E. F. Friedl. Mastering regular expressions: powerful techniques for Perl and other tools. Beijing, China, O'Reilly, 1998.

[9] A. Griewank; D. Juedes; H. Mitev; J. Utke; O. Vogel; A. Walther. Algorithm 755: Adol-C: a package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software* **22**, p. 131–167, 1996.

[10] J. D. Hamilton. CGI programming 101: programming Perl for the World Wide Web. Houston, EUA, CGI101.com, 1999.

[11] P. Wainwright. Professional Apache. Birmingham, Reino Unido, Wrox, 2000.