UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2569

ARCABOUÇO COMPUTACIONAL PARA O AUXÍLIO DO DESENVOLVIMENTO DE ROTINAS EXTERNAS C DENTRO DO AMBIENTE SCILAB

ANDERSON GONÇALVES MARCO
ODEMIR MARTINEZ BRUNO

Nº 315

RELATÓRIOS TÉCNICOS

São Carlos – SP
Jan./2008

Resumo

Linguagens e ambientes específicos para a computação científica, auxiliam o pesquisador em tarefas que envolvem computação e matemática. Entretanto a escrita de rotinas para estas ferramentas e feita em geral utilizando uma linguagem interpretada não apresentando deste modo um desempenho ótimo. O objetivo deste trabalho, é apresentar um arcabouço computacional para favorecer a integração da linguagem C com Scilab (Gomez, 1999), fazendo uma interoperabilidade do ambiente Scilab com o depurador da GNU (GDB). Esta integração permite utilizar os benefícios do ambiente Scilab e o desempenho da linguagem C.

Índice

1. Introdução

Plataformas científicas computacionais são ambientes (programas), que disponibilizam uma série funções matemáticas tais como transformadas matemáticas, plotamento de gráficos e afins.

O uso de Plataformas científicas computacionais para resolução discreta de problemas matemáticos físicos, biológicos e afins, tem se tornado muito comum recentemente, graças a popularidade que os computadores conquistaram nas ultimas décadas. Como estes ambientes de modelagem matemática são usados em sua maioria por pessoas que não possuem a programação de computadores como atividade principal em sua vida profissional, estes ambientes fazem uso de uma linguagem de programação de alto nível e geralmente interpretada, para criação de rotinas definidas pelo usuário. Pelas características (mencionadas acima) das linguagens utilizadas por estes ambientes, elas não apresentam um bom desempenho quando comparadas com linguagens como C e Fortran, que são linguagens de programação de mais baixo nível (consequentemente mais difícil de se programar) e que apresentam um menor numero de recursos matemáticos quando comparadas com estes ambiente.

Para sanar este problema muitas plataformas cientificas disponibilizam uma integração de suas linguagens com linguagens de mais baixo nível (podendo deste modo criar rotinas em linguagens como C e Fortran que rodem dentro destes ambientes). Contudo a criação de rotinas nestas linguagens, mostra-se ser difícil, no caso do ambiente Scilab que foi o analisado neste trabalho. Um dos principais problemas foi a depuração de rotinas escritas C, pelo fato de que este ambiente não possui uma ferramenta para a depuração de rotinas escritas em linguagem C.

Para sanar o problema de depuração de rotinas C dentro do ambiente Scilab, foi desenvolvido um arcabouço computacional para auxiliar no desenvolvimento e depuração de rotinas externas escritas em C. Na sessão 2 deste relatório e apresentado o ambiente Scilab e as diferentes maneiras de se escrever rotinas externas para ele, na sessão 3 e mostrada a implementação da transformada discreta Fourier em uma rotina externa escrita em linguagem C e em outra rotina escrita em linguagem Scilab,sendo feita uma comparação de velocidade entre elas no final da sessão, na sessão 4 e feita uma introdução a o uso de depuradores,mostrando os diferentes tipos de funcionamento deles, dando uma abordam em especial para o depurador GDB que foi o utilizado para elaboração do arcabouço, na sessão 5 e mostrado o funcionamento do arcabouço computacional criado pelo autor desse relatório, junto com um fluxograma,para facilitar o entendimento da funcionamento do arcabouço, ao final da sessão, na sessão 6 e mostrada uma implementação do arcabouço utilizando as linguagens python e C.

2. O Ambiente Scilab

O Scilab é uma plataforma científica voltado para computação numérica. Criado em 1990 foi mantido por pesquisadores pertencentes ao Institut de Recherche em Informatique et en Automatique, INRIA ate Maio de 2003, atualmente é mantido e desenvolvido pelo Consorcio Scilab.

Suas principais características são:

1. É um software de distribuição gratuita, com código fonte disponível. Sua linguagem é simples e de fácil aprendizado;
2. Possui um sistema de auxílio ao usuário, help;
3. É um ambiente poderoso para geração de gráficos bi-dimensionais e tri-dimensionais, inclusive com animação;
4. Implementa diversas funções para manipulação de matrizes.
5. Permite trabalhar com polinômios, sistemas lineares e grafos;
6. Apresenta facilidades para a definição de funções;
7. Permite o acesso a rotinas escritas nas linguagens FORTRAN ou C;
8. Pode ser acessado por programas de computação simbólica como o Maple1, que é um software comercial, ou o MuPAD2, que é livre para uso em instituições de ensino/pesquisa;
9. Suporta o desenvolvimento de conjuntos de funções voltadas para aplicações específicas, os chamados toolboxes.

Como mencionado acima, o Scilab permite que rotinas ou funções escritos em FORTRAN ou C sejam utilizados dentro de seu ambiente. Nesta seção, é apresentado os procedimentos necessários para ligação do Scilab com funções escritas na linguagem C.

Uma função escrita na linguagem C pode ser ligada ao Scilab de três maneiras distintas:

- através do comando link, em um processo chamado de ligação dinâmica;

- através de programas de interface, os chamados gateways, escritos pelo usuário ou gerados pelo intersi.

• Ou através da adição de uma nova função ao código do Scilab.

Neste trabalho apenas as duas primeiras maneiras serão apresentadas.

2.1 Ligação dinâmica

E realizado por meio do comando link que liga o código objeto,arquivo compilado mas não transformado em executável, a uma função indicada pelo usuário através de um parâmetro do comando link(este parâmetro e uma string).

Exemplo:

```
-->link('foo.o','foo','c');
linking files runge.o to create a shared executable
shared archive loaded
Linking foo
Link done
ans =
0.
-->a=[1,2,3;4,5,6];b= %pi;
-->[m,n]=size(a);
-->c=call("foo",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");
//Entrada
//a     matriz de entrada.
//      2 indica qual argumento da função(do arquivo objeto foo.o)
//recebera o endereço de memória da variável a

//b     variável de entrada.
//      3 indica qual argumento da função(do arquivo objeto foo.o)
//recebera o endereço de memória da variável b


//m     colunas da matriz de entrada a.
//      4 indica qual argumento da função(do arquivo objeto foo.o)
//recebera o endereço de memória da variável m

//n     linhas da matriz de entrada ª
//      4 indica qual argumento da função(do arquivo objeto foo.o)
//recebera o endereço de memória da variável n

//"out" indica que as informações adiante são referentes a o valor
//retornado pelo comando call.
```

```
//      [m,n] dimensão da matriz de saída retornada por call.
//      1 indica qual argumento da função(do arquivo objeto foo.o)
//retornado pelo comando call.

//c variável  que recebera o valor retornado por call.
```

```
#include <math.h>
void foo(c,a,b,m,n)/*Declaração dos parâmetros da  função*/
double a[],*b,c[]; /*Definição dos tipos de cada parâmetro da função*/
int *m,*n; /*Definição dos tipos de cada parâmetro da função*/
/* c[] Vetor que terá seus valores retornados pelo comando call */
{
        int i;
        for ( i =0 ; i < (*m)*(*n) ; i++)
                c[i] = sin(a[i]) + *b;
}
```

Código 1: Código fonte do arquivo objeto foo.o

Como pode ser visto no código 1,do exemplo, a função do nosso arquivo objeto foo.o não retorna valor algum oque sera retornado sera uma das variáveis que fazem parte dos parâmetros da função.

2.2 Interfaceando código C

Na sub-seção anterior foi visto como integrar C com Scilab por meio do comando call. Entretanto, este procedimento tem a desvantagem de utilizar o comando call para chamar as rotinas externas, oque difere da maneira usual utilizada nas funções do Scilab.

Existe no entanto uma maneira que permite chamar rotinas externas diretamente(da mesma forma que se faz com uma função nativa do Scilab). Isto pode ser feito por meio da criação de getways, que permite criar interfaces que fazem ligação da rotina externa com o Scilab. Getways são um conjunto de arquivos e funções,escritas em C,que podem ser feitos manualmente ou gerados automaticamente através do script intersci,eles permitem que se crie uma interface C

no Scilab, tornando a função C parte do Scilab,podendo assim chamar esta função,após o carregamento da interface, da mesma maneira que se chamaria uma função nativa do Scilab, neste trabalho será mostrada a criação deste arquivos de maneira automática.

O exemplo apresentado a seguir usa o gerador de interfaces intersci, que automatiza o processo de criação de vários arquivos necessários para construção de uma interface.

Exemplo:

Para este exemplo será utilizado o mesmo arquivo fonte C mostrado no exemplo anterior(foo.c),mas com algumas pequenas modificações. No mesmo diretório, deve ser criado um arquivo texto contendo duas descrições (separadas por uma linha em branco) da mesma função, da qual será gerada a interface, a primeira descrição será de como o Scilab interpreta função e a segunda e de como a rotina será de fato, esta descrição dirá ao intersci como criar a getway, necessária para criação da interface.

```
foo  a b
a      vector  m
b      vector  n
c      vector  m

foo  c a b m n
c      double
a      double
b      double
m      integer
n      integer

out    sequence       c
**********************
```

Código 2: Arquivo foo.desc, contendo as definições da função, que sera utilizado pelo intersci para criação da interface

Segue uma descrição de cada linha da definição do arquivo foo.desc:

foo a b: define o nome com o qual a rotina (identifica-da como foo) e seus parâmetros (determinados como a e b) serão conhecidos pelo ambiente Scilab.

a vector m : estabelece o parâmetro "a " como sendo um vetor de tamanho m.

b vector n : estabelece o parâmetro "b " como sendo um vetor de tamanho n.

c vector m : estabelece o parâmetro "c ",este parâmetro é oque sera retornado com a chamada do procedimento, como sendo um vetor de tamanho m.

Linha em branco: Separa a descrição de como o Scilab deve ver a procedure, da descrição de como é procedure.

foo c a b m n : define o nome com o qual a rotina (identifica-da como foo) e seus parâmetros (determinados como c a b m e n) estarão presentes.

c double: estabelece o parâmetro "c " como sendo um double.

a double: estabelece o parâmetro "a " como sendo um double.

b double: estabelece o parâmetro "b " como sendo um double.

m interge: estabelece o parâmetro "m " como sendo um int.

n integer: estabelece o parâmetro "n " como sendo um int.

Linha em branco: Separa a descrição de como é procedure do trecho em que e definido a variável a ser retornada pela chamada da interface.

out sequance c: estabelece o parâmetro "c" que sera retornado com a chamada da rotina.

```
#include "/usr/lib/Scilab-4.0/routines/machine.h"/*Necessária para criação
da interface*/
#include <math.h>
/*F2C converte os argumentos passados pelo Scilab em variáveis C que
são passadas para a função foo*/
void F2C(foo)(c,a,b,m,n)/*Declaração dos parâmetros da função*/
double a[],*b,c[]; /*Definição dos tipos de cada parâmetro da função*/
int *m,*n; /*Definição dos tipos de cada parâmetro da função*/
/* c[] Vetor que terá seus valores retornados pelo comando call */
{
        int i;
        for ( i =0 ; i < (*m)*(*n) ; i++)
                c[i] = sin(a[i]) + *b;
}
```

Código 3: Arquivo foo.c modifica para criação de uma interface Scilab

Usando o comando intersci -n "nome do arquivo" (colocando apenas o nome do arquivo sem sua extensão) e criado os arquivos necessários para compilação e criação da interface Scilab.

Se listar o diretório onde executou o comando intersci -n verá que foram criados uma serie de arquivos nesse diretório, estes arquivos permitem a criação da interface.

Para compilar e carregar a interface use o seguintes comando dentro do ambiente Scilab:

```
-->//Defina os valores dos vetores files e libs
-->//responsáveis por dizer os arquivos a serem compilados
-->//e quais bibliotecas devem ser linkadas.
-->files=['foo.o','"nome do arquivo".o'];
-->libs=[];//Inclui as libs necessárias para compilação da função
-->//Neste caso não foi necessário linkar nenhuma biblioteca
-->//  especial
-->exec "nome do arquivo"_builder.desc
-->exec loader.sce
-->//Pode-se chamar a rotina como qualquer outra função do Scilab
-->a=[2 3 4];
-->b=23
-->c=foo(a,b);
```

Como pode ser observado a criação de uma rotina externa mostra-se uma tarefa árdua, entretanto a velocidade de uma rotina externa é muito superior a velocidade de uma rotina escrita em linguagem Scilab. Como será mostrado na próxima seção.

3. Teste de velocidade das linguagem C e Scilab

O desempenho geral das duas linguagens foi testado implementando em ambas a Transformada Discreta de Fourier. Esta transformada e uma técnica clássica na área de processamento de sinais, sua principal característica é transformar uma função (sinal) no domínio do tempo para o domínio da frequência decompondo o sinal em senos e cossenos com diferentes frequências. Abaixo é visto a fórmula da Transformada Discreta de Fourier, os códigos fontes das implementações desta transformada na linguagem Scilab e em uma interface Scilab escrita na linguagem C e uma tabela contendo o tempo(em milesegundos) de processamento da transformada para ambas as implementações, para vetores com um determinado número de elementos.

$$x_k = \frac{1}{n}\sum_{j=0}^{n-1} f_j e^{\frac{2\pi i}{n}jk} \qquad k = 0, \ldots, n-1$$

Formula 1: Transformada discreta de Fourier

Sessão Editar Ver Favoritos Configurações Ajuda

```
function [transform]=fourier_trans(fx)
        n0=size(fx);
        N=n0(2);
        fu=zeros(1,N);
        fx1=zeros(1,N);
        fd1=zeros(1,N);
        cont=0;
        cont2=0;
        Preal=0;
        Pimag=0;
        while cont < N then
                Preal=0;
                Pimag=0;
                while cont2<N
                        Preal=((fx(cont2+1)*(cos((-2*%pi*cont*cont2)/N)))+Preal);
                        Pimag= ((fx(cont2+1)*(sin((-2*%pi*cont2*cont)/N)*%i))]+Pimag;
                        cont2=cont2+1;
                end
                fu(cont+1)=(Preal);
                fu(cont+1)=((Pimag)+fu(cont+1))/N;
                cont2=0;
                cont=cont+1;
        end
        transform=fu
endfunction
```

"fourier_codigo_scilab.sci" [dos] 26L, 504C written 26,0-1 All

Shell | Shell Núm. 2 | Shell Núm. 3

Figura 1: Função para a Transformada Discreta de Fourier escrita em script Scilab.

Figura 2: Rotina para a Transformada Discreta de Fourier escrita em linguagem C.

Transformada Discreta de Fourier

Nº de elementos no vetor	1	2	4	8	16	32	64	128	256	512	1024
Tempo em milisegundos Scilab	2	3	4	8	23	85	310	1221	4844	19373	72918
Tempo em milisegundos C	1	1	1	1	1	1	1	5	22	83	335

Tabela 1: Tempo decorrido para Transformada Discreta de Fourier em um Vetor.

Como se pode observar pelos dados da tabela 1 a implementação escrita em C apresenta um desempenho superior a implementação escrita em linguagem Scilab.

Entretanto,como já mencionado anteriormente, a criação de rotinas escritas em linguagem C mostrou-se mais trabalhosa que as rotinas escritas em script Scilab.

4. Depuradores uma maneira de facilitar a criação de código

Como mostrado na seção anterior a criação de rotinas externas para Scilab e um trabalho árduo, mas a velocidade de sua execução (quando comparada com uma rotina feita em Scilab) faz com que a criação de uma interface seja interessante, principalmente na solução de problemas computacionais que em se deseja desempenho. Uma maneira de facilitar (e diminuir o tempo) a criação de interface seria facilitar sua depuração, isto é facilitar localização e solução de eventuais erros que acontecem nas funções que definimos, ao longo desta seção será apresentado e discutido o estudo de depuradores para rotinas externas.

Um depurador, ou debugador, é um programa que facilita a localização e correção de erros. Praticamente todos as linguagens de programação tem um ou mais depuradores. Os depuradores podem ser divididos em dois tipos:

1. Integrados ao ambiente de desenvolvimento: São depuradores que vem junto com IDE,software para desenvolvimento de aplicações que integra compilador, editor de texto e depurador (lógico), esses deputares tem a vantagem de serem simples de usar entretanto não podem ser automatizados com scripts,arquivos em que pode-se definir regras de depuração, e sua integração com outras aplicações,tirando aquelas que são desenvolvidas pelos fabricante do IDE, e baixa.
   Exemplos desses depuradores são os dos IDES: Visual C++, Borland Delphi, Turbo Pascal e etc.

2. Depuradores independentes: São programas completos não fazendo parte de nenhum outro programa, mesmo assim eles podem,caso o IDE permita, se integrar a um ambiente de desenvolvimento,

geralmente podem depurar mais de uma linguagem de programação e podem ser automatizados com scripts.

Quase todos os depuradores desta categoria são, ou ao menos acompanham, os sistemas Unix. Exemplos deles são: GDB e DBX.

Apesar destas qualidades muitos programadores,acostumados a um ambiente IDE, geralmente programadores windows, dizem que estes depuradores são difíceis de se usar, mas os programadores,que não usam um ambiente de programação,em geral programadores Unix falam que estes debugadores apesar de mais complexos são mais práticos, uma vez que se tenha aprendido a usá-los.

Para este trabalho foi utilizado o depurador independente GDB, o principal a razão por esta escolha é por ele ser o depurador default de distribuições Linux e por possuir versões tanto pra Linux quanto Windows, fazendo deste modo o arcabouço portável. Utilizar o gdb é muito fácil, como será ilustrado:

- Para se depurar um programa no Gdb tudo que tem que fazer é compilar o programa no compilador gcc (GNU C Compiler) com opção menos -g, indicando desta forma ao compilador que queremos depurar o executável gerado, do seguinte modo "**gcc -g nome_do_arquivo_fonte.c -o saída**", feito isto, chame o gdb, via linha de comando, passando para ele como argumento o executável gerado,do seguinte modo "gdb saída". Com isto entramos no prompt do gdb onde se pode usar os seguintes comandos:
  1. **List** é usado para apresentar uma porção do código do programa. Normalmente pode-se chamar "list" (ou "l") seguido de um número que corresponderia a linha do código fonte. Também é permitido passar como argumento o nome de uma função ou o endereço de memória.
  2. **Break** é um ponto de parada usado para se marcar um ponto de parada numa linha no código. Normalmente pode-se chamar

"break" (ou "b") seguido de um número que corresponderia a linha do código fonte, cada ponto possui um número de identificação.

3. **Info breakpoints** é usado para se listar todos os pontos de parada que colocamos, seguidos de seus números de identificação. Normalmente utiliza-se chamar "info breakpoints" (ou "b i") .
4. **Del** é usado para se deletar pontos de parada. Normalmente pode-se chamar "del" (ou "d") seguido do numero de identificação do ponto de parada que se deseja deletar, caso nenhum número seja passado todos os pontos de parada serão excluídos.
5. **Print** é usado para mostrar a informação guardada numa variável quando o GDB atinge um ponto de parada. Normalmente pode-se chamar "print" (ou "p") seguido no nome da variável da qual queremos a informação.
6. **Next** é usado,depois que se atingiu um ponto de parada , para se executar a próxima linha a partir do ponto de parada, se usado novamente vai para a próxima depois desta e assim por adiante. Normalmente pode-se chamar "next" (ou "n").
7. **Continue** é usado para se ir ao próximo ponto de parada. Normalmente pode-se chamar "continue" (ou "c").

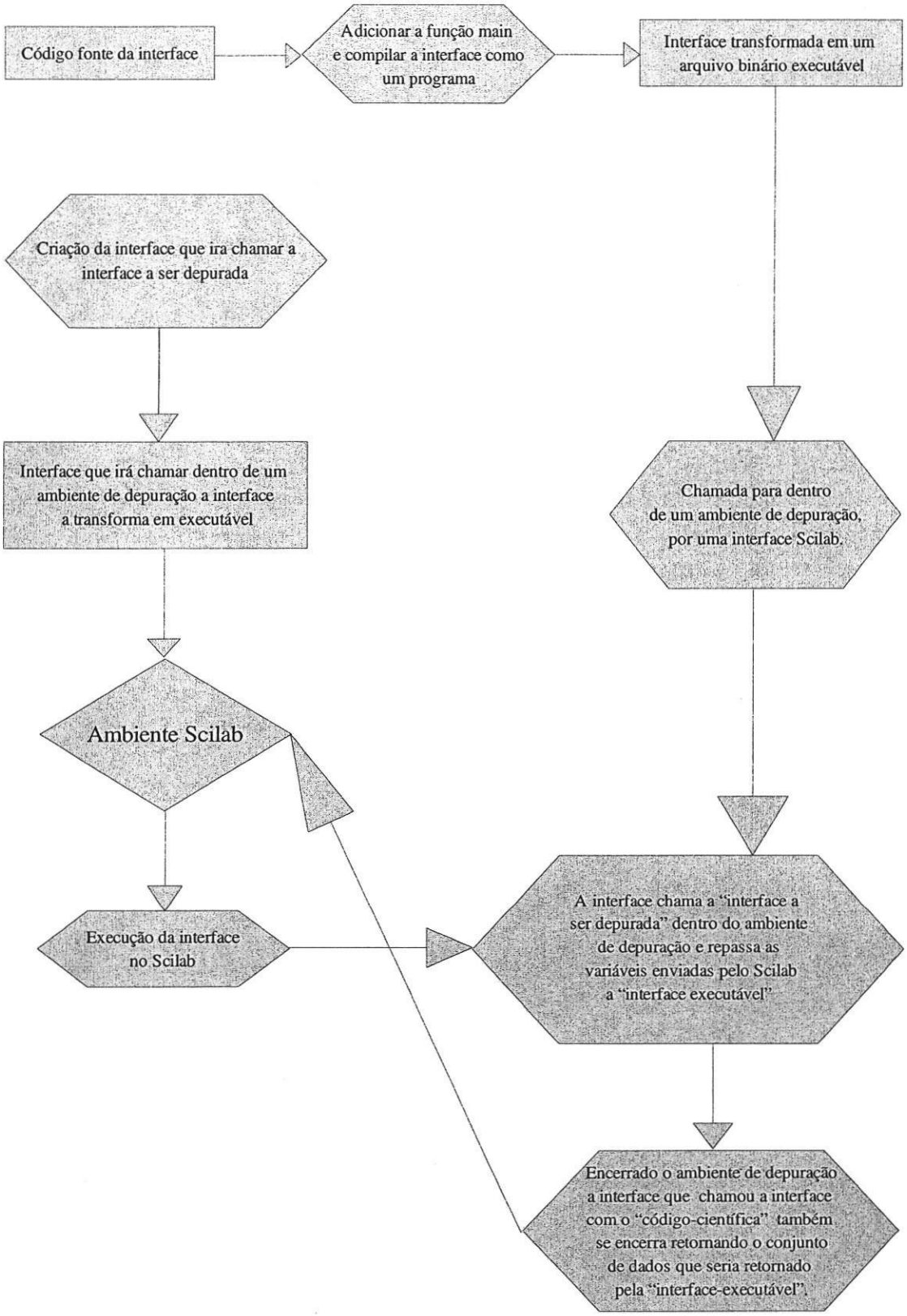
5. Criação do framework para depuração de interfaces externas

Como mostrado anteriormente a criação de rotinas externas para Scilab e um trabalho árduo, mas a velocidade de sua execução (quando comparada com uma rotina feita em Scilab) faz com que a criação de uma interface seja interessante, principalmente na solução de problemas computacionais em que se deseja desempenho. Deste modo, surge a necessidade de diminuir o tempo de criação uma interface.

Ao longo desta seção será apresentado um arcabouçou que visa diminuir o tempo para elaboração de uma interface scilab por meio da integração do scilab com a ferramenta de depuração gdb(ou seus frontends).

Para que se possa realizar a depuração de interfaces é necessário que a interface seja um arquivo executável, para isso se faz necessário que a mesma possua uma função **main** e que seja compilada como executável, entretanto quando essas modificações são feitas a interface deixa de ser uma interface scilab e passa a ser um programa executável, deste modo é preciso que esta "interface modificada" seja chamada por uma outra interface que não contenha nenhum "código-científico". Para que o programa de depuração seja chamada de dentro do Scilab, é preciso que interface chamada pelo Scilab envie as variáveis passadas para a interface chamada e retorne a matriz (ou vetor) que sera retornada da "interface modificada", para isto e necessário a criação de rotinas de comunicação.

O Fluxograma mostrado a seguir ilustra os passos para depuração de uma interface.


Código fonte da interface
Adicionar a função main e compilar a interface como um programa
Interface transformada em um arquivo binário executável
Criação da interface que ira chamar a interface a ser depurada
Interface que irá chamar dentro de um ambiente de depuração a interface a transforma em executável
Chamada para dentro de um ambiente de depuração, por uma interface Scilab.
Ambiente Scilab
Execução da interface no Scilab
A interface chama a "interface a ser depurada" dentro do ambiente de depuração e repassa as variáveis enviadas pelo Scilab a "interface executável"
Encerrado o ambiente de depuração a interface que chamou a interface com o "código-científica" também se encerra retornando o conjunto de dados que seria retornado pela "interface-executável".

6. Implementação do framework para depuração de interfaces externas

Na seção anterior foi descrito o funcionamento do arcabouço, que tem como objetivo depuração de rotinas externas C, nesta seção será abordado a utilização e a implementação do mesmo, para sua implementação foi escolhido as linguagens Python e C, a primeira foi empregada na construção e modificação dos arquivos necessários para depuração da interface e a segunda foi utilizada para criação das rotinas de comunicação entre as duas interfaces.
A seguir é mostrado como utilizar o arcabouço.

1. Crie dois arquivos fonte C, um com a função que será depurada e o outro vazio, (como mostrado na Figura 1) que será usado para criar a interface que será de fato chamada pelo Scilab.

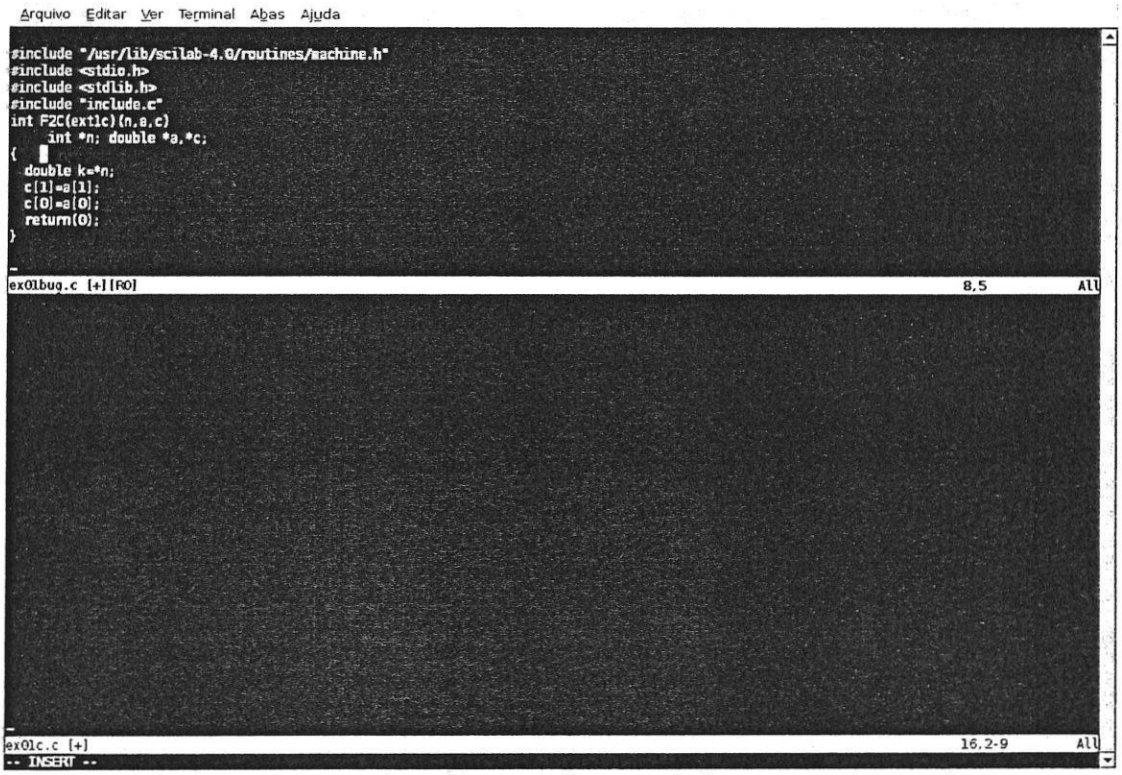


Figura 3: Imagem de 2 arquivos exemplos que foram criados para trabalhar com o arcabouço o de cima(ex01bug.c) e o arquivo que será depurado e o debaixo(ex01c.c) e oque será chamado pelo Scilab

2. Use o script **gerador_include.py**, este script irá criar arquivos texto, dentro do diretório dados, para que se possa haver a troca de variáveis entre os 2 arquivos C que foram criados na etapa 1, ele também gerará um script de depuração(script_debug) e um arquivo include.c que deve ser incluído dentro do arquivo C que será depurado. O arquivo include.c faz com que seja possível gerar um

executável da interface a ser depuratória utilizar o script use o seguinte comando: **gerador_include.py** <tipos de parâmetros da função> <"nome da função"> os tipos de parâmetros,devem ser colocados na ordem em que foram declarados pela função, eles podem ser:
d : para double

i : para inteiro

c : para strings

Caso a função esteja dentro da F2C é necessário colocar o parâmetro -F2C <nome da função> no final dos parâmetros passados para o script.
Exemplo de uso do script para o nosso exemplo da figura 1:

python gerador_include.py i d d "F2C(ext1c)" -F2C ext1c

3. Agora é preciso passar os valores das variáveis passadas como argumento da interface chamada pelo Scilab(exe1c.c) para interface que estamos depurando(ex1bug.c), para isto crie um script intersci, e rode o script **sci_debug_interface-1.py** passando como argumento para ele um script intersci, com isto será escrito no arquivo em branco mostrado na figura 1, o ex01c.c, o código,em C, da interface que chamará de dentro do Scilab a interface que desejamos depurar(a qual conterá o código cientifico).

4. Agora compile a interface a ser depurada com o seguinte comando "**gcc -g arquivo_fonte.c -o arquivo_fonte**", depois compile a interface criada no passo anterior, e a chame-a de dentro do Scilab, você verá que será aberta uma janela parecida com a da figura 2.

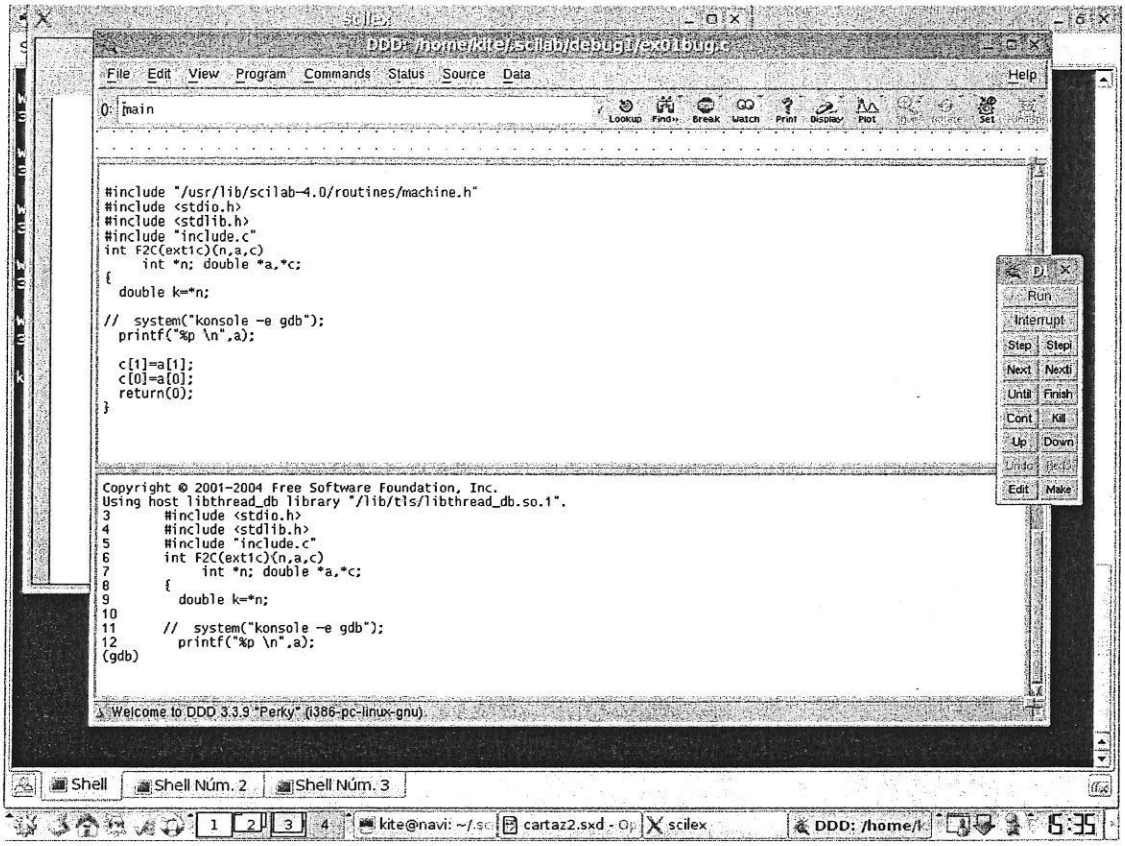


Figura 4: Imagem da execução da depuração de uma interface chamada de dentro do scilab, o depurador ajuda foi o front end do gdb, o DDD

A seguir o código fonte dos arquivos que compõe o framework.

```
#include <stdio.h>
#include <string.h>
/*Módulo responsável pelo envio de dados da interface chamada pelo Scilab  para arquivos temporários no HD*/

int n_um_scilab=1;/*Id do dado a ser enviado*/

int env_dados(void *val, char tipo,int tam){
        /*void *val-> recebe um vetor de dados a serem enviado*/
        /*char *tipo-> informa o tipo dos dados a serem enviados*/
        /*char *tipo-> informa o tamanho do vetor val*/
        FILE *arq;
        char cam[200];/*Buffer responsável por guardar a localização dos dados a serem transferidos, da interface chamada pelo Scilab para o HD*/

        int cont=0;/*Variável contadora*/
        if(tipo=='c'){/*Código responsável para envio de dados do tipo char*/
                char *dado=val;
                sprintf(cam,"dados/dado000%d.c",n_um_scilab);/*especifica a
        localização e nome do arquivo no qual o vetor sera gravado, a extensão *.c indica se o vetor é do tipo char*/

                arq=fopen(cam,"w");
                fprintf(arq,"%s",dado);/*grava os dados(existentes no vetor) em arquivo texto*/

                fclose(arq);
        }
        if(tipo=='d'){/*Codigo responsável para envio de dados do tipo double*/
                double *dado=val;
                sprintf(cam,"dados/dado000%d.d",n_um_scilab);/*especifica a
        localização e nome do arquivo no qual o vetor sera gravado, a extensão *.d indica e o vetor é do tipo double*/
                arq=fopen(cam,"w");
                while(cont<(tam-1)){
                        fprintf(arq,"%f\n",*dado);/*grava os dados(existentes no
                vetor) em arquivo texto*/
                        cont++;
                        dado++;
                }
                fprintf(arq,"%f",*dado);/*grava o último dado,existente no vetor,em
        arquivo texto*/
                fclose(arq);
        }
        if(tipo=='i'){/*Codigo responsável para envio de dados do tipo int*/
                int *dado=val;
                sprintf(cam,"dados/dado000%d.i",n_um_scilab);
```

```
                arq=fopen(cam,"w");
                while(cont<(tam-1)){
                        fprintf(arq,"%d\n",*dado);/*especifica a localização e  onome do
                arquivo no qual o vetor sera gravado, a extensão *.i indica e o vetor é do tipo
int*/
                        cont++;
                        dado++;
                }
                fprintf(arq,"%d",*dado);/*grava o ultimo dado,existente no  vetor,em arquivo
texto*/
                fclose(arq);
        }
        n_um_scilab++;/*incrementa a variável de id em +1, para que se saiba a ordem com que
os vetores foram enviados*/
 }
```

Arquivo: env_dados.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/*Módulo responsavel pelo envio de dados dos arquivos temporários no HD
para interface chamada pelo Scilab*/
void rec_dados(int num,char tipo,void * val){
        /*int num-> informa o id do dado a ser recebido*/
        /*char *tipo-> informa o tipo dos dados a serem recebidos*/
        /*void *val-> recebe um vetor de dados a serem recebidos*/

        FILE *arq;
        char cam[2000];/*Buffer responsável por guardar a localização do vetor,de dados, a ser
transferido, do arquivo do HD para a interface chamada pelo Scilab*/

        char tmp[2000];/*Buffer responsável por guardar a representação na forma string dos
dados pegos no arquivo temporário*/

        int cont=0;
        if(tipo=='c'){/*Codigo responsável para recebimento de dados do tipo        char*/

                char *dado=val;
                sprintf(cam,"dados/dado000%d.c",num);/*especifica a localização e nome do
arquivo no qual sera lido o vetor, a extensão *.c indica e o vetor e do tipo char*/

                arq=fopen(cam,"r");
                while(!feof(arq)){
                        fscanf(arq,"%s",dado);/*Lê os dados existentes no arquivo
        temporário e os passa para o vetor*/

                        dado++;
                }
                fclose(arq);
        }
        if(tipo=='d'){/*Código responsável para recebimento de dados do tipo        double*/

                double *dado=val;
                sprintf(cam,"dados/dado000%d.d",num);
                arq=fopen(cam,"r");
                while(!feof(arq)){/*Lê os dados existentes no arquivo temporário e passa para o
vetor*/

                        fgets(tmp,1998,arq);/*Lê o dado existente no arquivo
                temporário e os passa para o Buffer tmp*/

                        *dado=atof(tmp);/*Devolve a representação no formato
                double,do dado armazenado no buffer, guardando esta
```

```
        representação numa posição do vetor*/

                dado++;
                }
                fclose(arq);
        }
        if(tipo=='i'){/*Código responsável para recebimento de dados do tipo int*/

                int *dado=val;
                sprintf(cam,"dados/dado000%d.i",num);
                arq=fopen(cam,"r");
                while(!feof(arq)){/*Lê os dados existentes no arquivo temporário e os passas
para o vetor*/

                        fgets(tmp,1998,arq);/*Lê o dado existente no arquivo temporário e os
passa para o Buffer tmp*/

                        *dado=atoi(tmp);/*Devolve a representação no formato int, do dado
armazenado no buffer, guardando esta representação numa posição do vetor*/

                        dado++;
                }
                fclose(arq);
        }
}
```

Arquivo: rec_dados.c

```
#Script para criação dos arquivo include.c
import os, sys
func=sys.argv[1:]
if((len(func))>0):#Verifica se foi passado algum argumento pra o script
        tam=len(lista)
        cont=0
        cont1=0
        while((func[cont1]=="d") or (func[cont1]=="i") or (func[cont1]=="c")):#Cria os
arquivos que trocam dados entre as interfaces, com base nos argumentos passados para o script
                if(func[cont1]=="d"):#Caso o dado seja um double
                        temp="dados/dado000"+str(cont1+1)+".d"
                        inc=open(temp, "w")
                        inc.write("0")
                        inc.close()
                if(func[cont1]=="i"):#Caso seja um inteiro
                        inc=open("dados/dado000"+str(cont1+1)+".i", "w")
                        inc.write("0")
                        inc.close()
                if(func[cont1]=="c"):#Caso seja um String
                        inc=open("dados/dado000"+str(cont1+1)+".c", "w")
                        inc.write("0")
                        inc.close()
                cont1=cont1+1

        lista=os.listdir("dados")#Olha quantos arquivos exitem no diretório dados, cada arquivo
                                 é um vetor de dados, e cria uma lista com o nome deles

        #Inicia a criação do arquivo include.c
        pri="#include <stdlib.h>\n"
        pri=pri+"#include <string.h>\n"
        pri=pri+"#include <stdio.h>\n"
        pri=pri+"int main(void){\n"
        pri=pri+"       int cont, cont_dou, tam;\n"
        pri=pri+"       FILE *arq;\n"
        pri=pri+"       char dado[2000];\n"
        pri=pri+"       dado[1999]=0;\n"
        for arq in lista:#Loop responsável pela criação das linhas necessárias para recebimento
                        de dados,dos arquivos no hd, cada vetor a ser recebido precisa do
                        conjunto de linhas existentes no Loop
                pri=pri+"       int num"+str(cont)+"=0;\n" #Linha que contem a variável que
                                                        guarda o numero de elementos do
                                                        vetor
                pri=pri+"       char cam"+str(cont)+"[]= \"dados/"+arq+"\";\n"#Buffer que i
                                                                        indica a
                                                                        localização do
                                                                        dado
```

```
pri=pri+"	arq=fopen(cam"+str(cont)+",\"r\");\n"
pri=pri+"	while(!feof(arq)){\n"#Responsável por saber quantos elementos o
                                        vetor tem
pri=pri+"		num"+str(cont)+"++;\n"
pri=pri+"		fscanf(arq,\"%s\",dado);\n"
pri=pri+"	}\n"
pri=pri+"	fclose(arq);\n"
pri=pri+"	fopen(cam"+str(cont)+",\"r\");\n"
pri=pri+"	cont=0;\n"
if (arq[-1:]=="d"):#Caso o vetor seja do tipo double
        pri=pri+"	double dado"+str(cont)+"[num"+str(cont)+"];\n"#Cria um
                vetor com o tamanho igual a o numero de elementos do vetor
                salvo no arquivo
        pri=pri+"	while(!feof(arq)){\n"#Captura os elementos do arquivo é
                                        transfere para o vetor criado na
                                        linha acima
        pri=pri+"		fscanf(arq,\"%s\",dado);\n"
        pri=pri+"		dado"+str(cont)+"[cont]=atof(dado);\n"
        pri=pri+"		cont++;\n"
if (arq[-1:]=="i"):#Caso o vetor seja do tipo int
        pri=pri+"	int dado"+str(cont)+"[num"+str(cont)+"];\n"#Cria um
        vetor com o tamanho igual a o numero de elementos do vetor salvo no
        arquivo
        pri=pri+"	while(!feof(arq)){\n"
        pri=pri+"		fscanf(arq,\"%s\",dado);\n"
        pri=pri+"		dado"+str(cont)+"[cont]=atoi(dado);\n"
        pri=pri+"		cont++;\n"

if (arq[-1:]=="c"):#Caso o vetor seja do tipo string
        pri=pri+"	char dado"+str(cont)+"[num"+str(cont)+"][200];\n"#Cria
        um vetor com o tamanho igual a o numero de elementos do vetor salvo
        no arquivo
        pri=pri+"	while(!feof(arq)){\n"#Captura os elementos do arquivo é
        transfere para o vetor criado na linha acima
        pri=pri+"		fscanf(arq,\"%s\",dado);\n"
        pri=pri+"		strcpy(dado"+str(cont)+"[cont],(dado));\n"
        pri=pri+"		cont++;\n"

pri=pri+"	}\n"
pri=pri+"	fclose(arq);\n"
cont=cont+1

    pri=pri+"/*----------Chamada da interface Scilab que desejamos
depurar---------- */\n"
    pri=pri+"	"+func[0+cont1]+"("#Chamada da interface Scilab que desejamos
                                        depurar
```

```
        cont=0
        while(cont<tam):#Passa a para a interface que desejamos depurar os vetores capturados
                        do hd
                pri=pri+"&dado"+str(cont)+"[0],"
                cont=cont+1
        pri=pri[:-1]
        pri=pri+");\n"
        arq=" "
        cont=0
        for arq in lista:#Conjunto de linhas responsável por enviar os dados que foram
                processados pela interface chamada acima, após a depuração, para os respectivos
                arquivos dos quais os dados, existentes nos vetores, foram extraídos Cada vetor a
                ser recebido precisa do conjunto de linhas existentes no Loop
                pri=pri+"       arq=fopen(cam"+str(cont)+",\"w\");\n"#Abre o arquivo do qual
                foram extraídos os dados que estavam no vetor
                pri=pri+"       cont=0;\n"
                pri=pri+"       while(cont<num"+str(cont)+"){\n"
                if (arq[-1:]=="d"):#Caso o vetor seja do tipo double
                        pri=pri+"
        fprintf(arq,\"%f0.30\\n\",dado"+str(cont)+"[cont]);\n"
                if (arq[-1:]=="i"):#Caso o vetor seja do tipo int
                        pri=pri+"               fprintf(arq,\"%d\\n\",dado"+str(cont)+"[cont]);\n"
                if (arq[-1:]=="c"):#Caso o vetor seja tipo string
                        pri=pri+"               fprintf(arq,\"%s\\n\",dado"+str(cont)+"[cont]);\n"
                pri=pri+"               cont++;\n"
                pri=pri+"       }\n"
                pri=pri+"       fclose(arq);\n"
                cont=cont+1
        pri=pri+"}\n"
        inc=open("include.c", "w")#Grava a string pri no arquivo include.c
        inc.write(pri)
        inc.close()

else: #Caso o parâmetros passados para função tenham sido passados de forma errada
        print "Informe o nome da função passando-a como parâmetro do script o nome da
        função deve estar entre aspas\(simples ou duplas\)"
        print "ex:"
        print "  python gerador_include.py \"nome_da_função\""
        print "ou:"
        print "  python gerador_include.py d i d i \"nome_da_função\""

if (len(func)>cont1):#Gera um script de depuração,utilizado pelo GDB, a ser usado quando a
                        interface chamada pelo Scilab chamar a interface que desejamos depurar
        if(len(func)>(cont1+1)):
                if (func[cont1+1]=="-F2C"):
                        script="list "+func[cont1+2]+"_"
```

```
                print script
                inc=open("script_debug", "w")
                inc.write(script)
                inc.close()
        else:
                script=func[cont1+2]
                inc=open("script_debug", "w")
                inc.write(script)
                inc.close()
```

Arquivo: gerador_include.py

```
import os, sys, string
#sci_debug_interface
#Script Python que gera uma interface que chama a interface que desejamos depurar em
depuradores como o GDB, ou em seus frontends como o ddd
#Script criado por Anderson Gonçalves Marco

func=sys.argv[1:] #Argumentos passados para o script

pri=""
cond0=False #Serve para saber se os argumentos passados via console foram 3
cond=0 #Serve para saber se o arquivo passado como argumento para o console realmente
existe
cond2=True # Serve para escrever os includes da nossa interface de depuração
cond3=True #Para saber quando se deve ler as linhas do arquivo passado como argumento via
console
cond4=True #Para saber em que lugar se deve escrever a declaração da função
cond5=True #Para declaração dos  parâmetros da função de depuração
cond6=False #Para declaração dos  parâmetros da função quando um de seus parâmetros é uma
variável complexa
cond7=False #Para saber quando chegamos na penúltima linha do arquivo que contem a
definição da interface
posi=0
strlis0=[] #Passa saber os tipos de dados dos argumentos que foram passados do Scilab para C
strlis1=[] #Passa saber os parâmetros que nossa função de interface C possui
linhaf=[] #Para saber o valor que sera retornado pela nossa interface Scilab
cam_arq="" #Caminho do arquivo em que sera guardado esta interface
if((len(func))==3):
        cond0=True
        cond=os.path.exists(func[0])

if(cond and cond0):#Verifica se os parâmetros foram passados corretamente e se o arquivo do
                qual sera gerada interface existe
        arq=open(func[0], "r")
        lista=arq.readlines()
        arq.close()

        for linha in lista:
                if(cond2):
                        pri=pri+"#include \"/usr/lib/Scilab-4.0/routines/machine.h\"\n"
                        pri=pri+"#include <stdlib.h>\n"
                        pri=pri+"#include <stdlib.h>\n"
                        pri=pri+"#include \"env_dados.c\"\n"
                        pri=pri+"#include \"rec_dados.c\"\n"
                else:
                        temp=linha.split();
                        if(len(temp)<>0):
```

```
                    if(cond7):
                        linhaf=temp
                        cond7=False
                    if(cond3):#Serve para dividir palavras de uma linha do arquivo
                        em um lista (o critério para dividir uma linha e uma lista
                        de palavras e o espaço em branco)
                        strlis0.append(temp)
                    else:
                        if(cond4==False):
                            if(cond5):#Declara os tipos dos parâmetros da
                                função de interface
                                strlis1.append(temp)#Guarda os
                                                parâmetros que
                                                nossa interface
                                                possui eles serão
                                                usados mais tarde
                                                quando formos
                                                passá-los para
                                                interface que
                                                estamos depurando
                                temp3=[]
                                tam=0
                                for temp3 in strlis0:#Verifica se o dado é
                                                ou não do tipo
                                                complexo se sim
                                                (cond6=True) ou
                                                se não
                                                (Cond6=False)
                                    if(temp3[1]=="imatrix" and
temp3[0]==temp[0]):
                                        tam=len(temp3)
                                        cond6=True
                                        break
                                cond7=True

                                if(cond6):# Dado do tipo complexo
                                    pri=pri+"int     *"+temp3[tam-
1]+"r"+temp[0]+";\n"#Declara um inteiro que diz se o dado é inteiramente real ou não
                                    pri=pri+"double   *"+temp[0]+"r,
*"+temp[0]+"i"+";\n" #Declara parte imaginaria e real do dado
                                    cond6=False
                                else:# Não é do tipo complexo o dado
                                    if(temp[1]=="double"):
                                        pri=pri+"double
*"+temp[0]+";\n"
                                    if(temp[1]=="integer"):
```

```
                                                    pri=pri+"int
*"+temp[0]+";\n"
                                    if(cond4):#Declara a função da interface e seus
                                            parâmetros(que não possuem tipo ainda)
                                            pri=pri+"F2C("+temp[0]+") ("
                                            cam_arq=temp[0]
                                            temp2=""
                                            temp3=[]
                                            tam=0
                                            for temp2 in temp[1:]:#Verifica se o dado é ou não
                                                    do tipo complexo se
                                                    sim(cond6=True) ou se não
                                                    (Cond6=False)
                                                for temp3 in strlis0:
                                                    if(temp3[1]=="imatrix" and
temp3[0]==temp2):
                                                    tam=len(temp3)
                                                    cond6=True
                                                    break
                                                if(cond6):# É do tipo complexo o dado
                                                pri=pri+temp2+"r,"+temp2+"i,"+temp3
[tam-1]+"r"+temp2+"," #Declara  as variáveis de que o tipo de dado complexo é feito
                                    cond6=False
                                                else:# Não e do tipo complexo o dado
                                                    pri=pri+temp2+","
                                            pri=pri[:-1]+");\n"
                                            cond4=False

                            else:
                                    if(cond3==False):
                                            cond5=False
                                    cond3=False
                    cond2=False

            #Definição da função
            pri=pri+"{\n"
            cont=0
            cont2=0
            temp=[]
            for temp in strlis1:#Envia os dados dessa interface para a interface que desejamos
                            depurar para isso usa a função env_dados do arquivo env_dados.c
                            cond6=False
                    if(temp[0]==linhaf[2]):
                            posi=cont2+1
                    if(temp[1]=="integer"):#Verifica se o dado e um inteiro
                            pri=pri+"       env_dados(&"+temp[0]+"[0],i,1);\n"
```

```
                if(temp[1]=="double"):#Verifica se o dado e um double
                        temp3=[]
                        temp4=[]
                        temp2=""
                        tam=0
                        for temp3 in strlis0:#Verifica se o dado é ou não do tipo complexo se
                                                sim(cond6=True) ou se não (Cond6=False)
                                if(temp3[1]=="imatrix" and temp3[0]==temp[0]):
                                        tam=len(temp3)
                                        cond6=True
                                        break
                        if(cond6):# E do tipo complexo o dado
                                pri=pri+"       env_dados(&"+temp[0]+"r[0],d,"      #envia parte
real                            do dado
                                temp4=strlis0[cont]
                                        #************************
                                for temp2 in temp4[2:-1]:
                                        #************************
                                        pri=pri+temp2+"*"
                                                #************************
                                pri=pri[:-1]+");\n"
                                        #************************

                                pri=pri+"       env_dados(&"+temp[0]+"i[0],d,"      #envia parte
                                                                                    imaginaria
                                                                                    do dado
                                cont2=cont2+1
                                        #*******************************
                                temp4=strlis0[cont]
                                        #*******************************
                                for temp2 in temp4[2:-1]:
                                        #*******************************
                                        pri=pri+temp2+"*"
                                                #*******************************
                                pri=pri[:-1]+");\n"
                                        #*******************************

                                pri=pri+"       env_dados(&"+temp3[tam-
1]+"r"+temp[0]+"[0],i,1);\n"    #Envia um inteiro que diz se o dado é inteiramente real ou não
                                cont2=cont2+1
                        else:   #Não é do tipo complexo o dado
                                pri=pri+"       env_dados(&"+temp[0]+"[0],d,"
                                temp4=strlis0[cont]
                                for temp2 in temp4[2:]:
                                        pri=pri+temp2+"*"
                                pri=pri[:-1]+");\n"
```

```
                    cont=cont+1
                    cont2=cont2+1
        temp2=[]
        temp=""
        pri=pri+"       system(\""+func[1]+" "+func[2] +" --command script_debug\");\n" #
        Chama a interface que desejamos depurar
        for temp2 in strlis0:
            if(temp2[0]==linhaf[2]):
                if(temp2[1]=="imatrix"):
                    pri=pri+"       rec_dados("+str(posi+1)+",d,"+temp2[0]+"i);\n"
                        #recebe parte real do dado
                    pri=pri+"       rec_dados("+str(posi+2)+",d,"+temp2[0]+"r);\n"
                        #recebe parte imaginaria do dado
                    pri=pri+"       rec_dados("+str(posi+3)+",i,"+temp2[-
1:][0]+"r"+temp2[0]+");\n"#recebe e um inteiro que diz se o dado é inteiramente real ou não
                else:   #Não é do tipo complexo o dado
                    pri=pri+"       =rec_dados("+str(posi+1)+",d,"+temp2[0]+");\n"
        pri=pri+"       n_um_Scilab=1;\n"
        pri=pri+"       return(0);\n"
        pri=pri+"}\n"
        if(os.path.exists(cam_arq)):#Verifica se já não existe um arquivo que tem o mesmo
nome do arquivo em que interface sera salva
            while(True):
                print "O arquivo com nome:"+cam_arq
                print "já existe você deseja continuar:[sim/não]"
                resp=raw_input()
                if(resp=="sim"):
                    arq=open(cam_arq, "w")
                    inc.write(pri)
                    inc.close()
                    arq=open(cam_arq, "w")
                    inc.write(pri)
                    inc.close()
                    break
                if(resp=="não"):
                    break

else:
    if(cond0):
        print "O arquivo usado para gerar a interface não existe de depuração"
    else:
        print "Numero de parâmetros passados para a função estão incorretos"
```

Arquivo:sci_debug_interface-1.py

7. Conclusão

Neste trabalho foi visto uma das principais plataformas cientificas, o Scilab, mostrando como se criar rotinas externas ,escritas em C. Para facilitar a criação de rotinas escritas em linguagem C, foi criado um arcabouçou capaz de integrar o depurador GDB, com a plataforma cientifica Scilab.

8. Bibliografia

(Gomez,) Gomez, Claude (Ed.) Engineering and Scientific Computing with Scilab, Birkhäuser book, 1999

(Stallman et al,) Debugging with GDB: The GNU Source-Level Debugger. Richard M. Stallman, Roland H. Pesch, Stan Shebs, GNU Press, 2000.

(Brad Dayley,) Python Phrasebook, Sams Publishing, 2006

(Herbert Schildt,) C COMPLETO E TOTAL, PEARSON EDUCATION DO BRASIL LTDA, 1996