Trabalho 1 - CI1164 - Introdução a Computação Científica

Aluno, Matheus Pacheco dos Santos - GRR20197286, login dinf: mps19@inf.ufpr.br

Aluna, Luzia Millena Santos Silva - GRR20197286, login dinf: lmss18@inf.ufpr.br

---

Vamos documentar o nosso trabalho. Vamos descrever cada `.c`. Mas antes valem ressaltar que funções que fazem alocações ou usam funções que fazem, retornam `-1` caso a alocação falhe. Inclusive é o único código de erro possível nas funções programadas.

utils.c

Estas descrições de funções podem ser vistas no próprio `utils.h`

```
/*  Retorna tempo em milisegundos

Forma de uso:

double tempo;
tempo = timestamp();
<trecho de programa do qual se deseja medir tempo>
tempo = timestamp() - tempo;
*/

double timestamp(void);
```

alloc.c

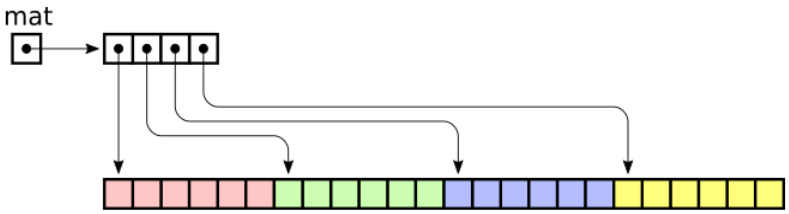
Estas descrições de funções podem ser vistas no próprio `allloc.h`

```
/* Aloca matriz de doubles em mat
* Retorna 0 se houve sucesso, caso contrário retorna -1 representando falha de alocação - única falha possível
*/
int alocaMatDoubles(int lin, int col, double ***mat);

/* Aloca matriz de chars em mat
* Retorna 0 se houve sucesso, caso contrário retorna -1 representando falha de alocação - única falha possível
*/
int alocaMatChars(int lin, int col, char ***mat);

/* Aloca matrizs de void* em mat
* Retorna 0 se houve sucesso, caso contrário retorna -1 representando falha de alocação - única falha possível
*/
int alocaMatPonteirosVoids(int lin, int col, void ****mat);

/* Limpa memória alocada para a matriz mat */
void freeMatChars(char ***mat);

/* Limpa memória alocada para a matriz mat */
void freeMatDoubles(double ***mat);

/* Limpa memória alocada para a matriz mat */
void freeMatPonteirosVoids(void ****mat, int lin, int col);
```

Vamos exemplificar como a função `int alocaMatDoubles(int lin, int col, double ***mat);` como é feito a alocação desses tipos. O código da função é:

```c
int alocaMatDoubles(int lin, int col, double ***mat) {
    int i;

    // aloca um vetor de lin ponteiros para linhas
    (*mat) = malloc(lin * sizeof (double*));
    if ((*mat) == NULL) {
        return -1;
    }

    // aloca um vetor com todos os doubles da matriz
    (*mat)[0] = malloc(lin * col * sizeof(double));
    if ((*mat)[0] == NULL) {
        free((*mat));
        return -1;
    }

    // ajusta os demais ponteiros de linhas (i > 0)
    for (i=1; i < lin; i++) {
        (*mat)[i] = (*mat) [0] + i * col ;
    }

    return 0;
}
```

a ideia é criar um vetor de linhas contíguas, basicamenete é criar um vetor de ponteiros cujo cada ponteiro aponto pro início de cada vetor de elementos. Veja abaixo uma ilustração retirada do website do professor Maziero, o link da imagem pode ser encontrado aqui:



Todas alocações seguem esse padrão!

sistemas.c

Essa lib é a responsável pela resolução dos sistemas. As descrições das funções podem ser econtradas no `sistemas.h`

```c
/* Estrutura que respresenta um SNL. As funções são objetos evaluators da
    lib mathval */
typedef struct
{
    int numFuncoes;
    void **funcoes;
} SNL;

/* Estrutura que representa um SL na de forma que os coeficentes são
    representadis por uma matriz quadrática. Ou seja,
necessáriamente o SL deve ter o mesmo número de equações de variáveis
*/
typedef struct
{
    int dimensao;
    double **coeficientes;
    double *termosLivres;
```

```
16 } MatQuadraticaSL;
17
18 /* Seta funções e numFuncoes */
19 void inicializaSNL(SNL *sistema, void ***funcoes, int numFuncoes);
20
21
22 /* Libera memória alocada dinâmicamente */
23 void finalizaSNL(SNL *sistema);
24
25
26 /* Aloca espaço para a matriz de acordo com a dimensão. Utiliza a lib
       alloc.h
27 * Retorna 0 se houve sucesso, caso contrário retorna -1 representando
       falha de alocação - única falha possível
28 */
29 int inicializaMatQuad(MatQuadraticaSL *mat, int dimensao);
30
31
32 /* Libera memória alocada dinâmicamente */
33 void finalizaMatQuad(MatQuadraticaSL *mat);
34
35
36 /* Gera matriz de derivdas parciais do SNL sistema e armazena em mat
37 * mat deve ser previamente alocado
38 */
39 void geraMatDerivParcial(void ****mat, SNL sistema);
40
41 /* Realiza retro-substituição do SL representado por mat e armazena o
       resultado em x
42 * A matriz de coefecientes deve triangular inferior
43 */
44 void retrossubs(MatQuadraticaSL mat, double **x);
45
46
47 /* Retorna o índice da linha cujo o elemento da coluna j seja o maior
       dentre as colunas das linhas abaixo da linha j
48 * Deve-se tomar cuidado pois a linha não deve ser a última, pois se a
       linha for a última, i é o próprio pivo
49 */
50 int econtraMaxPivo(double ***mat, int j, int n);
51
52
53 /* Troca a linha i pela linha j do SL representado por mat
54 * Deve-se tomar cuidado para i não ser igual a j
55 */
56 void trocaLinhaMat(MatQuadraticaSL *mat, int i, int pivo);
57
58
59 /* Realiza eliminação atráves do metódo Gauss Jordan com pivoteamento
       parcial */
60 void eliminacaoGaussJordan(MatQuadraticaSL *mat);
61
62
63 /* Calcula matriz jacobiana de acordo com  a matDeriVParcial e valores de
       x e armazena em matJacobiana
64 * É necessário para o cálculo o numero de funcoes (numFuncoes) e  um vetor
        de variaveis da equação, da forma: ["x1","x2","x3"..."xn"]
65 * matJacobiana deve ser previamente alocada
66 */
67 void calculaMatJacobiana(void ***matDerivParcial, double ***matJacobiana,
       char **vars, int numFuncoes, double *valoresX);
```

```
68
69  /* Gera variaveis de acordo com a dimensão e armazena em vars
70  * Por exemplo, dimensao = 3, teremos: ["x1", "x2", "x3"]
71  * vars deve previamente alocado
72  */
73  void geraVars(char ***vars, int dimensao);
74
75
76  /* Print os tempos do metódo do newton, na sequencia: total, cálculo das
        derivadas,
77  cálculo das jacobianas e cálculo das resoluçõe dos SLs
78  */
79  void printaTemposMetodoNewtonSNL(FILE *saida, double tempos[4]);
80
81
82  /* Resolve o sistema não linear através do metódo de iteração newton
83  * Usa epsilon e maxIteracoes como critérios de parada. Se um dos dois
        forem atingidos o método encerra
84  * Printa no arquivo "saida" as aproximações e o tempo de execução total,
        tempo do cálculo das jacobianas e da resolução do SNL
85  * Retorna 0 se houve sucesso, caso contrário retorna -1 representando
        falha de alocação - única falha possível
86  */
87  int metodoNewtonSNL(SNL sistema, double *xAprox, double epsilon, int
        maxIteracoes, FILE *saida);
```

Vale resaltar que para as funções: `calculaMatJacobiana` e `geraMatDerivParcial` as matrizes onde serão aramazenadas a derivada e jacobiana devem ser alocadas previamente. Enquanto a função `inicializaMatQuad` fara uma alocação de `mat.coeficientes` dentro da própria função.

As funções: `retrossubs` e `eliminacaoGaussJordan` foram feitas com base nos algoritmos apresentados em aula.

Vamos exemplificar como é feito para calcular as funções usando a lib `matheval`. Para tal veja o código:

```
1  void calculaMatJacobiana(void ***matDerivParcial, double ***matJacobiana,
       char **vars, int numFuncoes, double *valoresX) {
2      for (int i = 0; i < numFuncoes; i++) {
3          for (int j = 0; j < numFuncoes; j++) {
4              (*matJacobiana)[i][j] = evaluator_evaluate(matDerivParcial[i][
                   j], numFuncoes, vars, valoresX);
5          }
6      }
7  }
```

é usado a função `evaluator_evaluate` da `matheval`, mas para isso é preciso um vetor de variáveis da função. Por exemplo considere a função $x1 + x2$, então temos que ter um vetor `vars` dessa forma: [”$x1$”,”$x2$”]. É feito dessa forma também o cálculo para a norma de funções inbutida na função `metodoNewtonSNL`. Esta função de gerar variáveis e feita por:

```
1  void geraVars(char ***vars, int dimensao) {
2      for (int i = 0; i <  dimensao; i++) {
3          sprintf((*vars)[i] , "x%d", i + 1);
4      }
5  }
```

A função do `metodoNewtonSNL` gera a matriz de derivadas parciais, aplica as iterações do metódo newton, calcula os tempos de execução total do metódo, cálculo das derivadas, cálculo da jacobianas e cálculo da resoluções dos SLs. Os tempos são impressos usando a função:

```
1  void printaTemposMetodoNewtonSNL(FILE *saida, double tempos[4]) {
2      fprintf(saida, "############\n");
3      fprintf(saida, "# Tempo Total: %lf\n", tempos[INDICE_TOTAL]);
4      fprintf(saida, "# Tempo Derivada: %lf\n", tempos[INDICE_DERIVADAS]);
```

```
5       fprintf(saida, "# Tempo Jacobiana: %lf\n", tempos[INDICE_JACOBIANA]);
6       fprintf(saida, "# Tempo SL: %lf\n", tempos[INDICE_SL]);
7       fprintf(saida, "############\n");
8   }
```

usando os indíces definidos no `sistemas.h`:

```
1   #define INDICE_TOTAL 0
2   #define INDICE_DERIVADAS 1
3   #define INDICE_JACOBIANA 2
4   #define INDICE_SL 3
```

Logo as funções `metodoNewtonSNL` e `printaTemposMetodoNewtonSNL` utilizam desas definições para manipular um vetor de tempos.

newtonSLN.c

Este `.c` contém a main do nosso código. Ele basicamente

- Le bloco de funções de acordo com a especificação do trabalho e chama função de resolução de SNLs;
- Imprime erros em stderr, e imprime os blocos de resolução de acordo com os argumentos -o <local de impressao> se a não for especificado imprime em stdout.