```c
#include <sys/time.h>
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#include "../includes/libpoisson.h"

/* Função que retorna o tempo em função do relógio */
double walltime( double *t0 )
{

    double mic, time;
    double mega = 0.000001;
    struct timeval tp;
    struct timezone tzp;
    static long base_sec = 0;
    static long base_usec = 0;

    (void) gettimeofday(&tp,&tzp);

    if (base_sec == 0)
    {
        base_sec = tp.tv_sec;
        base_usec = tp.tv_usec;
    }

    time = (double) (tp.tv_sec - base_sec);
    mic = (double) (tp.tv_usec - base_usec);
    time = (time + mic * mega) - *t0;
    return(time);
}

void errorMsg(char error_text[])
/* Standard error handler */
{
    fprintf(stderr,"Run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

/*Função que aloca as matrizes de nós, como arrays, de tamanho linXcol*/
nodeSides* criaVetorNode(int lin,int col)
{
    nodeSides* v_aux;

    v_aux = (nodeSides*) malloc(lin*col*sizeof(nodeSides));

    if(v_aux==NULL)
        errorMsg("allocation failure in vector");

    return v_aux;
}

/*Função que aloca as matrizes de nós de tamanho linXcol*/
nodeSides** criaMatrizNode(int lin,int col, nodeSides* v_aux)
{
    int i;
    nodeSides** p_aux;

    p_aux = (nodeSides**) malloc(lin*sizeof(nodeSides*));

    if(p_aux==NULL)
        errorMsg("allocation failure in vector");

    for(i=0; i<lin; i++)
        p_aux[i] = (nodeSides*) &v_aux[i*col];

    return p_aux;
}
```

```c

/*Função que aloca as matrizes de nós de tamanho linXcol*/
nodeSides** montaMatrizNode(int lin,int col)
{
    int i;
    nodeSides** p_aux;

    p_aux = (nodeSides**) malloc(lin*sizeof(nodeSides*));

    if(p_aux==NULL)
        errorMsg("allocation failure in vector");

    for(i=0; i<lin; i++)
    {
        p_aux[i] = (nodeSides*) malloc(col*sizeof(nodeSides));
        if(p_aux[i]==NULL)
            errorMsg("allocation failure in vector");
    }

    return p_aux;
}

/*Função que aloca as matrizes, como arrays, de tamanho linXcol*/
double* criaVetor(int lin,int col)
{
    double* v_aux;

    v_aux = (double*) malloc(lin*col*sizeof(double));

    if(v_aux==NULL)
        errorMsg("allocation failure in vector");

    return v_aux;

}

/*Função que aloca as matrizes de tamanho linXcol*/
double** criaMatriz(int lin,int col, double* v_aux)
{
    int i;
    double** p_aux;

    p_aux = (double**) malloc(lin*sizeof(double*));

    if(p_aux==NULL)
        errorMsg("allocation failure in vector");

    for(i=0; i<lin; i++)
        p_aux[i] = (double*) &v_aux[i*col];

    return p_aux;
}

/*Função que aloca as matrizes de tamanho linXcol*/
double** montaMatriz(int lin,int col)
{
    int i;
    double** p_aux;

    p_aux = (double**) malloc(lin*sizeof(double*));

    if(p_aux==NULL)
        errorMsg("allocation failure in vector");

    for(i=0; i<lin; i++)
    {
        p_aux[i] = (double*) malloc(col*sizeof(double));
        if(p_aux[i]==NULL)
            errorMsg("allocation failure in vector");
    }
```

```c

    return p_aux;
}

/*Função que aloca a matriz de parametros materiais, como
 *arrays, de tamanho linXcol
 */
nodeMaterial* criaVetorMaterial(int lin,int col)
{
    nodeMaterial* v_aux;

    v_aux = (nodeMaterial*) malloc(lin*col*sizeof(nodeMaterial));

    if(v_aux==NULL)
        errorMsg("allocation failure in dvector()");

    return v_aux;
}

/*Função que aloca a matriz de parametros materiais
 *de tamanho linXcol
 */
nodeMaterial** criaMatrizMaterial(int lin,int col, nodeMaterial* v_aux)
{
    int i;
    nodeMaterial** p_aux;

    p_aux = (nodeMaterial**) malloc(lin*sizeof(nodeMaterial*));

    if(p_aux==NULL)
        errorMsg("allocation failure in dvector()");

    for(i=0; i<lin; i++)
        p_aux[i] = (nodeMaterial*) &v_aux[i*col];

    return p_aux;
}

/*Função que aloca a matriz de parametros materiais
 *de tamanho linXcol
 */
nodeMaterial** montaMatrizMaterial(int lin,int col)
{
    int i;
    nodeMaterial** p_aux;

    p_aux = (nodeMaterial**) malloc(lin*sizeof(nodeMaterial*));

    if(p_aux==NULL)
        errorMsg("allocation failure in dvector()");

    for(i=0; i<lin; i++)
    {
        p_aux[i] = (nodeMaterial*) malloc(col*sizeof(nodeMaterial));
        if(p_aux[i]==NULL)
            errorMsg("allocation failure in vector");
    }

    return p_aux;
}

/* Função para o canto inferior esquerdo*/
void canto_d_l(const int i, const int j,
               nodeMaterial **pMat,
               nodeSides **beta,
               nodeSides **q,
               nodeSides **q_old,
               nodeSides **l_old,
               double **p)

```

```
{
    register double shi, AuxU,AuxR,DU,DR;       /* Auxiliares para cada lado das células */

    AuxU = pMat[i][j].shi/(1+beta[i][j].up*pMat[i][j].shi);
    AuxR = pMat[i][j].shi/(1+beta[i][j].rh*pMat[i][j].shi);
    DU = AuxU*(beta[i][j].up*q_old[i][j+1].dn+l_old[i][j+1].dn);
    DR = AuxR*(beta[i][j].rh*q_old[i+1][j].lf+l_old[i+1][j].lf);
    shi = (pMat[i][j].f + DU + DR)/(AuxU + AuxR);
    q[i][j].up = AuxU*shi - DU;
    q[i][j].rh = AuxR*shi - DR;
    p[i][j] = shi;
}

/* Função para o canto inferior esquerdo*/
void canto_d_lArray(const int i, const int j, const int N,
               nodeMaterial *pMat,
               nodeSides *beta,
               nodeSides *q,
               nodeSides *q_old,
               nodeSides *l_old,
               double *p)

{
    register double shi, AuxU,AuxR,DU,DR;       /* Auxiliares para cada lado das células */
    register int k = i*N + j;

    shi = pMat[k].shi;
    AuxU = shi/(1+beta[k].up*shi);
    AuxR = shi/(1+beta[k].rh*shi);
    DU = AuxU*(beta[k].up*q_old[k+1].dn+l_old[k+1].dn);
    DR = AuxR*(beta[k].rh*q_old[k+N].lf+l_old[k+N].lf);
    p[k] = shi = (pMat[k].f + DU + DR)/(AuxU + AuxR);
    q[k].up = AuxU*shi - DU;
    q[k].rh = AuxR*shi - DR;
}

/* Função para o canto superior esquerdo*/
void canto_u_l(const int i, const int j,    nodeMaterial **pMat,
               nodeSides **beta,
               nodeSides **q,
               nodeSides **q_old,
               nodeSides **l_old,
               double **p)
{
    register double shi,AuxD, AuxR,DD, DR;      /* Auxiliares para cada lado das células */

    AuxD = pMat[i][j].shi/(1+beta[i][j].dn*pMat[i][j].shi);
    AuxR = pMat[i][j].shi/(1+beta[i][j].rh*pMat[i][j].shi);
    DD = AuxD*(beta[i][j].dn*q_old[i][j-1].up+l_old[i][j-1].up);
    DR = AuxR*(beta[i][j].rh*q_old[i+1][j].lf+l_old[i+1][j].lf);
    shi = (pMat[i][j].f + DD + DR)/(AuxD + AuxR);
    q[i][j].dn = AuxD*shi - DD;
    q[i][j].rh = AuxR*shi - DR;
    p[i][j] = shi;
}

/* Função para o canto superior esquerdo*/
void canto_u_lArray(const int i, const int j, const int N,
                    nodeMaterial *pMat,
                    nodeSides *beta,
                    nodeSides *q,
                    nodeSides *q_old,
                    nodeSides *l_old,
                    double *p)
{
    register double shi,AuxD, AuxR,DD, DR;      /* Auxiliares para cada lado das células */
```

```
    register int k = i*N + j;

    shi = pMat[k].shi;
    AuxD = shi/(1+beta[k].dn*shi);
    AuxR = shi/(1+beta[k].rh*shi);
    DD = AuxD*(beta[k].dn*q_old[k-1].up+l_old[k-1].up);
    DR = AuxR*(beta[k].rh*q_old[k+N].lf+l_old[k+N].lf);
    p[k] = shi = (pMat[k].f + DD + DR)/(AuxD + AuxR);
    q[k].dn = AuxD*shi - DD;
    q[k].rh = AuxR*shi - DR;
}

/* Função para o canto inferior direito*/
void canto_d_r(const int i, const int j,    nodeMaterial **pMat,
               nodeSides **beta,
               nodeSides **q,
               nodeSides **q_old,
               nodeSides **l_old,
               double **p)
{

    register double shi, AuxU, AuxL,DU,DL;      /* Auxiliares para cada lado das células */

    AuxU = pMat[i][j].shi/(1+beta[i][j].up*pMat[i][j].shi);
    AuxL = pMat[i][j].shi/(1+beta[i][j].lf*pMat[i][j].shi);
    DU = AuxU*(beta[i][j].up*q_old[i][j+1].dn+l_old[i][j+1].dn);
    DL = AuxL*(beta[i][j].lf*q_old[i-1][j].rh+l_old[i-1][j].rh);
    shi = (pMat[i][j].f + DU + DL)/(AuxU + AuxL);
    q[i][j].up = AuxU*shi - DU;
    q[i][j].lf = AuxL*shi - DL;
    p[i][j] = shi;
}

/* Função para o canto inferior direito*/
void canto_d_rArray(const int i, const int j, const int N,
                    nodeMaterial *pMat,
                    nodeSides *beta,
                    nodeSides *q,
                    nodeSides *q_old,
                    nodeSides *l_old,
                    double *p)
{

    register double shi, AuxU, AuxL,DU,DL;      /* Auxiliares para cada lado das células */
    register int k = i*N + j;

    shi = pMat[k].shi;
    AuxU = shi/(1+beta[k].up*shi);
    AuxL = shi/(1+beta[k].lf*shi);
    DU = AuxU*(beta[k].up*q_old[k+1].dn+l_old[k+1].dn);
    DL = AuxL*(beta[k].lf*q_old[k-N].rh+l_old[k-N].rh);
    p[k] = shi = (pMat[k].f + DU + DL)/(AuxU + AuxL);
    q[k].up = AuxU*shi - DU;
    q[k].lf = AuxL*shi - DL;
}

/* Funcao para o canto superior dereito*/
void canto_u_r(const int i, const int j,    nodeMaterial **pMat,
               nodeSides **beta,
               nodeSides **q,
               nodeSides **q_old,
               nodeSides **l_old,
               double **p)
{

    register double shi, AuxD,AuxL,DD,DL;       /* Auxiliares para cada lado das células */

```

```c
    AuxD = pMat[i][j].shi/(1+beta[i][j].dn*pMat[i][j].shi);
    AuxL = pMat[i][j].shi/(1+beta[i][j].lf*pMat[i][j].shi);
    DD = AuxD*(beta[i][j].dn*q_old[i][j-1].up+l_old[i][j-1].up);
    DL = AuxL*(beta[i][j].lf*q_old[i-1][j].rh+l_old[i-1][j].rh);
    shi = (pMat[i][j].f + DD + DL)/(AuxD + AuxL);
    q[i][j].dn = AuxD*shi - DD;
    q[i][j].lf = AuxL*shi - DL;
    p[i][j] = shi;
}

/* Funcao para o canto superior dereito*/
void canto_u_rArray(const int i, const int j, const int N,
                    nodeMaterial *pMat,
                    nodeSides *beta,
                    nodeSides *q,
                    nodeSides *q_old,
                    nodeSides *l_old,
                    double *p)
{

    register double shi, AuxD,AuxL,DD,DL;       /* Auxiliares para cada lado das células */
    register int k = i*N + j;

    shi = pMat[k].shi;
    AuxD = shi/(1+beta[k].dn*shi);
    AuxL = shi/(1+beta[k].lf*shi);
    DD = AuxD*(beta[k].dn*q_old[k-1].up+l_old[k-1].up);
    DL = AuxL*(beta[k].lf*q_old[k-N].rh+l_old[k-N].rh);
    p[k] = shi = (pMat[k].f + DD + DL)/(AuxD + AuxL);
    q[k].dn = AuxD*shi - DD;
    q[k].lf = AuxL*shi - DL;
}


/* Função para a fronteira superior U */
void fronteira_u(const int n, const int j,
                 nodeMaterial **pMat,
                 nodeSides **beta,
                 nodeSides **q,
                 nodeSides **q_old,
                 nodeSides **l_old,
                 double **p)
{

    register double shi;
    register int i;
    register double AuxD, AuxR, AuxL, DD, DR, DL;       /* Auxiliares para cada lado das celulas */
    for (i = 2; i<n; i++)
    {
        shi = pMat[i][j].shi;
        AuxL = shi/(1+beta[i][j].lf*shi);
        AuxR = shi/(1+beta[i][j].rh*shi);
        AuxD = shi/(1+beta[i][j].dn*shi);
        DL = AuxL*(beta[i][j].lf*q_old[i-1][j].rh+l_old[i-1][j].rh);
        DR = AuxR*(beta[i][j].rh*q_old[i+1][j].lf+l_old[i+1][j].lf);
        DD = AuxD*(beta[i][j].dn*q_old[i][j-1].up+l_old[i][j-1].up);
        shi = (pMat[i][j].f + DD + DL + DR)/(AuxD + AuxL+AuxR);
        q[i][j].lf = AuxL*shi - DL;
        q[i][j].rh = AuxR*shi - DR;
        q[i][j].dn = AuxD*shi - DD;
        p[i][j] = shi;
    }

}

/* Função para a fronteira superior U */
void fronteira_uArray(const int N, const int j,
                      nodeMaterial *pMat,
```

```c
                      nodeSides *beta,
                      nodeSides *q,
                      nodeSides *q_old,
                      nodeSides *l_old,
                      double *p)
{

    register double shi;
    register int i, n = N*(N-2);
    register double AuxD, AuxR, AuxL, DD, DR, DL;       /* Auxiliares para cada lado das celulas */
    for (i=2*N+j; i < n; i+=N)
    {
        shi = pMat[i].shi;
        AuxL = shi/(1+beta[i].lf*shi);
        AuxR = shi/(1+beta[i].rh*shi);
        AuxD = shi/(1+beta[i].dn*shi);
        DL = AuxL*(beta[i].lf*q_old[i-N].rh+l_old[i-N].rh);
        DR = AuxR*(beta[i].rh*q_old[i+N].lf+l_old[i+N].lf);
        DD = AuxD*(beta[i].dn*q_old[i-1].up+l_old[i-1].up);
        p[i] = shi = (pMat[i].f + DD + DL + DR)/(AuxD + AuxL+AuxR);
        q[i].lf = AuxL*shi - DL;
        q[i].rh = AuxR*shi - DR;
        q[i].dn = AuxD*shi - DD;
    }

}

/* Função para a fronteira inferior D */
void fronteira_d(const int n, const int j,
                 nodeMaterial **pMat,
                 nodeSides **beta,
                 nodeSides **q,
                 nodeSides **q_old,
                 nodeSides **l_old,
                 double **p)
{
    register double shi;
    register int i;
    register double AuxU,AuxR, AuxL,DU,DR, DL;  /* Auxiliares para cada lado das celulas */
    for (i=2; i<n; i++)
    {
        shi = pMat[i][j].shi;
        AuxL = shi/(1+beta[i][j].lf*shi);
        AuxR = shi/(1+beta[i][j].rh*shi);
        AuxU = shi/(1+beta[i][j].up*shi);
        DL = AuxL*(beta[i][j].lf*q_old[i-1][j].rh+l_old[i-1][j].rh);
        DR = AuxR*(beta[i][j].rh*q_old[i+1][j].lf+l_old[i+1][j].lf);
        DU = AuxU*(beta[i][j].up*q_old[i][j+1].dn+l_old[i][j+1].dn);
        shi = (pMat[i][j].f + DU + DL + DR)/(AuxU + AuxL+AuxR);
        q[i][j].lf = AuxL*shi - DL;
        q[i][j].rh = AuxR*shi - DR;
        q[i][j].up = AuxU*shi - DU;
        p[i][j] = shi;
    }

}

/* Função para a fronteira inferior D */
void fronteira_dArray(const int N, const int j,
                      nodeMaterial *pMat,
                      nodeSides *beta,
                      nodeSides *q,
                      nodeSides *q_old,
                      nodeSides *l_old,
                      double *p)
{
    register double shi;
    register int i, n = N*(N-2);
```

```c
    register double AuxU,AuxR, AuxL,DU,DR, DL;  /* Auxiliares para cada lado das celulas
    */
    for (i=2*N+j; i < n; i+=N)
    {
        shi = pMat[i].shi;
        AuxL = shi/(1+beta[i].lf*shi);
        AuxR = shi/(1+beta[i].rh*shi);
        AuxU = shi/(1+beta[i].up*shi);
        DL = AuxL*(beta[i].lf*q_old[i-N].rh+l_old[i-N].rh);
        DR = AuxR*(beta[i].rh*q_old[i+N].lf+l_old[i+N].lf);
        DU = AuxU*(beta[i].up*q_old[i+1].dn+l_old[i+1].dn);
        p[i] = shi = (pMat[i].f + DU + DL + DR)/(AuxU + AuxL+AuxR);
        q[i].lf = AuxL*shi - DL;
        q[i].rh = AuxR*shi - DR;
        q[i].up = AuxU*shi - DU;
    }

}

/* Função para a fronteira dereita R */
void fronteira_r(const int i, const int n,
                 nodeMaterial **pMat,
                 nodeSides **beta,
                 nodeSides **q,
                 nodeSides **q_old,
                 nodeSides **l_old,
                 double **p)
{

    register double shi;
    register int j;
    register double AuxU,AuxD, AuxL,DU, DD, DL; /* Auxiliares para cada lado das celulas
    */
    for (j=2; j<n; j++)
    {
        shi = pMat[i][j].shi;
        AuxU = shi/(1+beta[i][j].up*shi);
        AuxD = shi/(1+beta[i][j].dn*shi);
        AuxL = shi/(1+beta[i][j].lf*shi);
        DU = AuxU*(beta[i][j].up*q_old[i][j+1].dn+l_old[i][j+1].dn);
        DD = AuxD*(beta[i][j].dn*q_old[i][j-1].up+l_old[i][j-1].up);
        DL = AuxL*(beta[i][j].lf*q_old[i-1][j].rh+l_old[i-1][j].rh);
        p[i][j] = shi = (pMat[i][j].f + DU + DL + DD)/(AuxU + AuxL+AuxD);
        q[i][j].up = AuxU*shi - DU;
        q[i][j].dn = AuxD*shi - DD;
        q[i][j].lf = AuxL*shi - DL;
    }

}

/* Função para a fronteira dereita R */
void fronteira_rArray(const int i, const int N,
                      nodeMaterial *pMat,
                      nodeSides *beta,
                      nodeSides *q,
                      nodeSides *q_old,
                      nodeSides *l_old,
                      double *p)
{

    register double shi;
    register int j, n = (i+1)*N - 2 ;;
    register double AuxU,AuxD, AuxL,DU, DD, DL; /* Auxiliares para cada lado das celulas
    */
    for (j=(i*N)+2; j < n; j++)
    {
        shi = pMat[j].shi;
        AuxU = shi/(1+beta[j].up*shi);
        AuxD = shi/(1+beta[j].dn*shi);
        AuxL = shi/(1+beta[j].lf*shi);
```

```
        DU = AuxU*(beta[j].up*q_old[j+1].dn+l_old[j+1].dn);
        DD = AuxD*(beta[j].dn*q_old[j-1].up+l_old[j-1].up);
        DL = AuxL*(beta[j].lf*q_old[j-N].rh+l_old[j-N].rh);
        p[j] = shi = (pMat[j].f + DU + DL + DD)/(AuxU + AuxL+AuxD);
        q[j].up = AuxU*shi - DU;
        q[j].dn = AuxD*shi - DD;
        q[j].lf = AuxL*shi - DL;
    }

}

/* Função para a fronteira esquerda L */
void fronteira_l(const int i, const int n,
                 nodeMaterial **pMat,
                 nodeSides **beta,
                 nodeSides **q,
                 nodeSides **q_old,
                 nodeSides **l_old,
                 double **p)
{
    register double shi;
    register int j;
    register double AuxU, AuxD, AuxR,DU, DD, DR; /* Auxiliares para cada lado das células
*/
    for (j=2; j<n; j++)
    {
        shi = pMat[i][j].shi;
        AuxU = shi/(1+beta[i][j].up*shi);
        AuxD = shi/(1+beta[i][j].dn*shi);
        AuxR = shi/(1+beta[i][j].rh*shi);
        DU = AuxU*(beta[i][j].up*q_old[i][j+1].dn+l_old[i][j+1].dn);
        DD = AuxD*(beta[i][j].dn*q_old[i][j-1].up+l_old[i][j-1].up);
        DR = AuxR*(beta[i][j].rh*q_old[i+1][j].lf+l_old[i+1][j].lf);
        shi = (pMat[i][j].f + DU + DR + DD)/(AuxU + AuxR+AuxD);
        q[i][j].up = AuxU*shi - DU;
        q[i][j].dn = AuxD*shi - DD;
        q[i][j].rh = AuxR*shi - DR;
        p[i][j] = shi;
    }

}

/* Função para a fronteira esquerda L */
void fronteira_lArray(const int i, const int N,
                      nodeMaterial *pMat,
                      nodeSides *beta,
                      nodeSides *q,
                      nodeSides *q_old,
                      nodeSides *l_old,
                      double *p)
{
    register double shi;
    register int j, n = (i+1)*N - 2 ;
    register double AuxU, AuxD, AuxR,DU, DD, DR; /* Auxiliares para cada lado das células
*/

    for (j=(i*N)+2; j < n; j++)
    {
        shi = pMat[j].shi;
        AuxU = shi/(1+beta[j].up*shi);
        AuxD = shi/(1+beta[j].dn*shi);
        AuxR = shi/(1+beta[j].rh*shi);
        DU = AuxU*(beta[j].up*q_old[j+1].dn+l_old[j+1].dn);
        DD = AuxD*(beta[j].dn*q_old[j-1].up+l_old[j-1].up);
        DR = AuxR*(beta[j].rh*q_old[j+N].lf+l_old[j+N].lf);
        p[j] = shi = (pMat[j].f + DU + DR + DD)/(AuxU + AuxR+AuxD);
        q[j].up = AuxU*shi - DU;
        q[j].dn = AuxD*shi - DD;
        q[j].rh = AuxR*shi - DR;
    }
```

```c
}

/* Função para os nós internos */
void internos(const int n,
              nodeMaterial **pMat,
              nodeSides **beta,
              nodeSides **q,
              nodeSides **q_old,
              nodeSides **l_old,
              double **p)
{
    register double shi;
    register int i,j;
    register double AuxU, AuxD, AuxR, AuxL, DU, DD, DR, DL; /* Auxiliares para cada lado das células */


    for (i=2; i<n; i++)
        for (j=2; j<n; j++)
        {
            shi = pMat[i][j].shi;
            AuxU = shi/(1+beta[i][j].up*shi);
            AuxR = shi/(1+beta[i][j].rh*shi);
            AuxD = shi/(1+beta[i][j].dn*shi);
            AuxL = shi/(1+beta[i][j].lf*shi);

            DL = AuxL*(beta[i][j].lf*q_old[i-1][j].rh+l_old[i-1][j].rh);
            DD = AuxD*(beta[i][j].dn*q_old[i][j-1].up+l_old[i][j-1].up);
            DR = AuxR*(beta[i][j].rh*q_old[i+1][j].lf+l_old[i+1][j].lf);
            DU = AuxU*(beta[i][j].up*q_old[i][j+1].dn+l_old[i][j+1].dn);

            shi = (pMat[i][j].f + DU + DD + DL + DR)/(AuxU + AuxD + AuxL+AuxR);

            q[i][j].up = AuxU*shi - DU;
            q[i][j].rh = AuxR*shi - DR;
            q[i][j].dn = AuxD*shi - DD;
            q[i][j].lf = AuxL*shi - DL;
            p[i][j] = shi;
        }

}

/* Função para os nós internos */
void internosArray(const int N,
                   nodeMaterial *pMat,
                   nodeSides *beta,
                   nodeSides *q,
                   nodeSides *q_old,
                   nodeSides *l_old,
                   double *p)
{
    register double shi;
    register int i, j,n = N-4;
    register double AuxU, AuxD, AuxR, AuxL, DU, DD, DR, DL; /* Auxiliares para cada lado das células */
    nodeSides *q_ant, *q_pos, *q_atu, *l_ant, *l_pos, *l_atu;
    nodeSides *beta_, *q_;
    double *p_;
    nodeMaterial *pMat_;

    q_ant = &q_old[1];
    q_atu = q_ant + N;
    q_pos = q_atu + N;

    l_ant = &l_old[1];
    l_atu = l_ant + N;
    l_pos = l_atu + N;

    pMat_ = &pMat[N+1];
    beta_ = &beta[N+1];
```

```c
    q_ = &q[N+1];
    p_ = &p[N+1];

    for (i=1; i <= n; i++)
    {
        q_ant += N;
        q_atu += N;
        q_pos += N;

        l_ant += N;
        l_atu += N;
        l_pos += N;

        pMat_ += N;
        beta_ += N;
        q_ += N;
        p_ += N;
        for (j=1; j <= n; j++)
        {
            shi = pMat_[j].shi;
            AuxU = shi/(1+beta_[j].up*shi);
            AuxR = shi/(1+beta_[j].rh*shi);
            AuxD = shi/(1+beta_[j].dn*shi);
            AuxL = shi/(1+beta_[j].lf*shi);

            DL = AuxL*(beta_[j].lf*q_ant[j].rh+l_ant[j].rh);
            DD = AuxD*(beta_[j].dn*q_atu[j-1].up+l_atu[j-1].up);
            DR = AuxR*(beta_[j].rh*q_pos[j].lf+l_pos[j].lf);
            DU = AuxU*(beta_[j].up*q_atu[j+1].dn+l_atu[j+1].dn);

            p_[j] = shi = (pMat_[j].f + DU + DD + DL + DR)/(AuxU + AuxD + AuxL+AuxR);

            q_[j].up = AuxU*shi - DU;
            q_[j].rh = AuxR*shi - DR;
            q_[j].dn = AuxD*shi - DD;
            q_[j].lf = AuxL*shi - DL;
        }
    }

}

/* Atualização dos multiplicadores de lagrange */
double lagrangeUpdate(const int n,
                      nodeSides **beta,
                      nodeSides **q,
                      nodeSides **q_old,
                      nodeSides **l,
                      nodeSides **l_old,
                      double **p)
{
    register double Media = 0.0;
    register int i,j;

    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
        {
            l[i][j].up = beta[i][j].up*(q[i][j].up + q_old[i][j+1].dn) + l_old[i][j+1].dn;
            l[i][j].dn = beta[i][j].dn*(q[i][j].dn + q_old[i][j-1].up) + l_old[i][j-1].up;
            l[i][j].rh = beta[i][j].rh*(q[i][j].rh + q_old[i+1][j].lf) + l_old[i+1][j].lf;
            l[i][j].lf = beta[i][j].lf*(q[i][j].lf + q_old[i-1][j].rh) + l_old[i-1][j].rh;
            Media += p[i][j];
        }

    Media /= (n*n);
    return Media;
}

/* Atualização dos multiplicadores de lagrange */
double lagrangeUpdateArray(const int N,
                           nodeSides *beta,
```

```
                            nodeSides *q,
                            nodeSides *q_old,
                            nodeSides *l,
                            nodeSides *l_old,
                            double *p)
{
    register int i,j, n = N-2;
    register double Media = 0.0;
    nodeSides *q_ant, *q_pos, *q_atu, *l_ant, *l_pos, *l_atu;
    nodeSides *beta_, *q_, *l_;
    double *p_;

    q_atu = &q_old[0];
    q_ant = q_atu - N;
    q_pos = q_atu + N;

    l_atu = &l_old[0];
    l_ant = l_atu - N;
    l_pos = l_atu + N;

    beta_ = &beta[0];
    q_ = &q[0];
    p_ = &p[0];
    l_ = &l[0];

    for (i=1; i <= n; i++)
    {
        q_ant += N;
        q_atu += N;
        q_pos += N;

        l_ant += N;
        l_atu += N;
        l_pos += N;

        beta_ += N;
        q_ += N;
        p_ += N;
        l_ += N;
        for (j=1; j<=n; j++)
        {
            l_[j].up = beta_[j].up*(q_[j].up + q_atu[j+1].dn) + l_atu[j+1].dn;
            l_[j].dn = beta_[j].dn*(q_[j].dn + q_atu[j-1].up) + l_atu[j-1].up;
            l_[j].rh = beta_[j].rh*(q_[j].rh + q_pos[j].lf) + l_pos[j].lf;
            l_[j].lf = beta_[j].lf*(q_[j].lf + q_ant[j].rh) + l_ant[j].rh;
            Media += p_[j];
        }
    }

    Media /= (n*n);
    return Media;
}

/* Impondo a média zero na distriubição de pressões
 * e cálculo de verificação de convergência */

double mediaZero(const int n,
                 double Media,
                 nodeSides **l,
                 double **p,
                 double **p_old)
{
    register double sum1, sum2, aux;
    register int i,j;
    sum1 = sum2 = 0.0;

    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
        {
            p[i][j] -= Media;
```

```
            l[i][j].up -= Media;
            l[i][j].dn -= Media;
            l[i][j].rh -= Media;
            l[i][j].lf -= Media;

            aux = p[i][j] - p_old[i][j];
            sum1 += aux*aux;
            sum2 += p[i][j] * p[i][j];
        }

    /*Erro relativo entre a pressão atual e anterior*/
    return sqrt(sum1/sum2);
}

/* Impondo a média zero na distriubição de pressões
 * e cálculo de verificação de convergência */

double mediaZeroArray(const int N,
                      double Media,
                      nodeSides *l,
                      double *p,
                      double *p_old)
{
    register int i,j, n = N-2;
    register double pj, sum1, sum2, aux;
    sum1 = sum2 = 0.0;
    nodeSides *l_;
    double *p_, *p__;

    p_ = &p[0];
    p__ = &p_old[0];
    l_ = &l[0];

    pj = Media;

    for (i=1; i <= n; i++)
    {
        p_ += N;
        p__ += N;
        l_ += N;
        for (j=1; j<=n; j++)
        {
            l_[j].up -= pj;
            l_[j].dn -= pj;
            l_[j].rh -= pj;
            l_[j].lf -= pj;
            pj = p_[j] -= pj;

            aux = pj - p__[j];
            sum1 += (aux*aux);
            sum2 += (pj*pj);
            pj = Media;
        }
    }

    /*Erro relativo entre a pressão atual e anterior*/
    return sqrt(sum1/sum2);
}
```