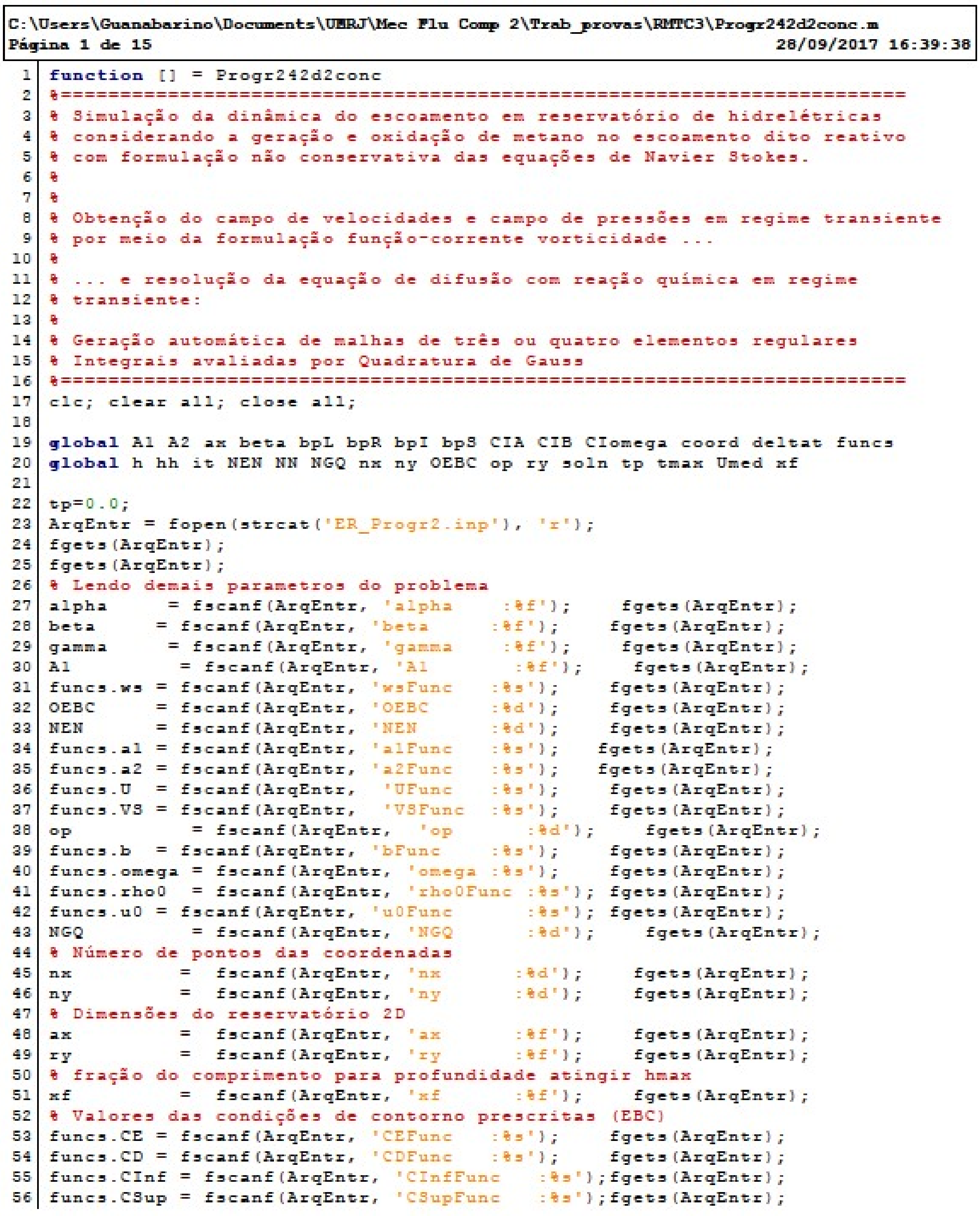
APPENDIX B – MATLAB CODE FOR COMPUTATION OF VELOCITY AND CONCENTRATION PROFILES

B.1 Source code



```
C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr242d2conc.m
Página 1 de 15                                                          28/09/2017 16:39:38
```

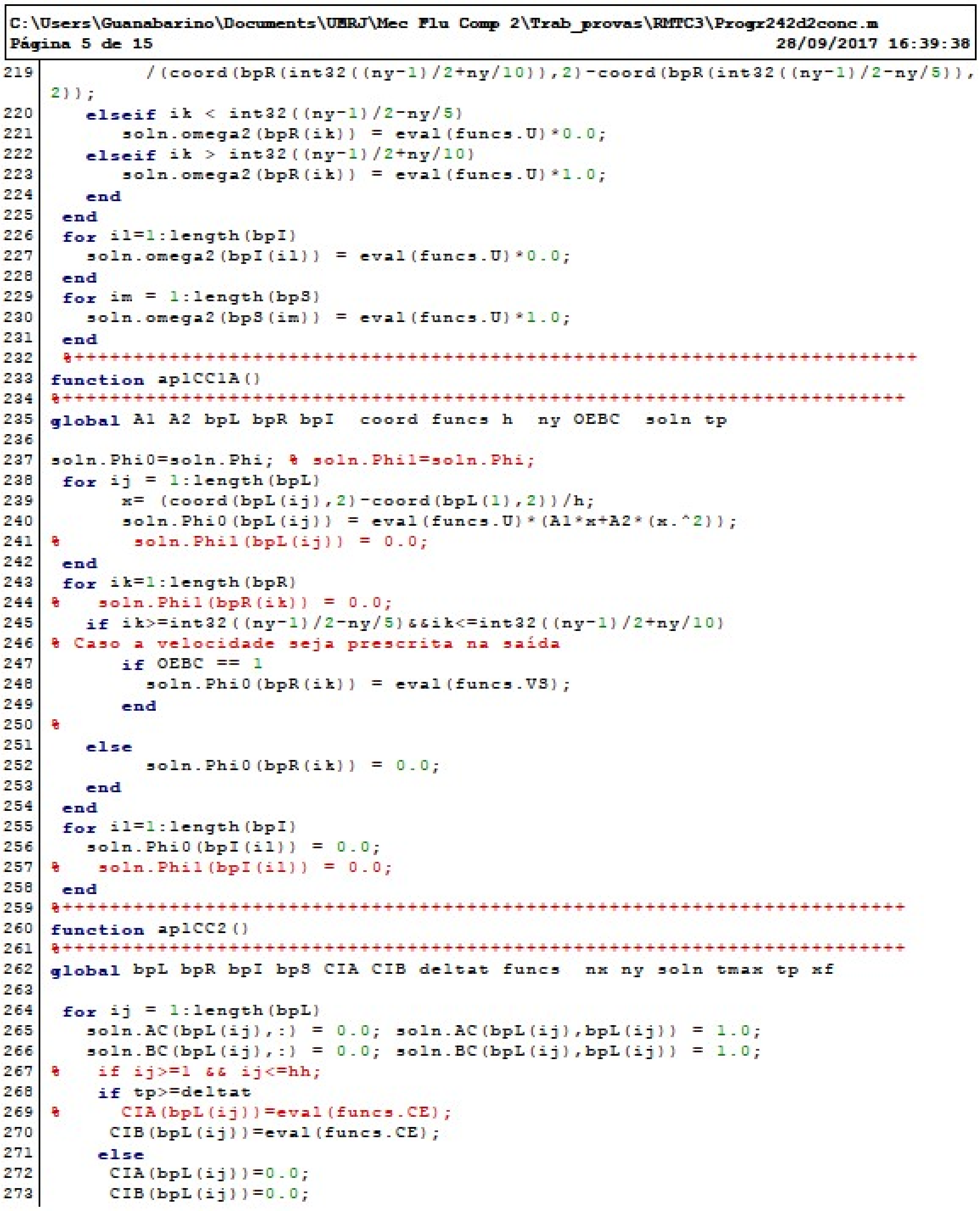
```matlab
function [] = Progr242d2conc
%========================================================================
% Simulação da dinâmica do escoamento em reservatório de hidrelétricas
% considerando a geração e oxidação de metano no escoamento dito reativo
% com formulação não conservativa das equações de Navier Stokes.
%
%
% Obtenção do campo de velocidades e campo de pressões em regime transiente
% por meio da formulação função-corrente vorticidade ...
%
% ... e resolução da equação de difusão com reação química em regime
% transiente:
%
% Geração automática de malhas de três ou quatro elementos regulares
% Integrais avaliadas por Quadratura de Gauss
%========================================================================
clc; clear all; close all;

global A1 A2 ax beta bpL bpR bpI bpS CIA CIB CIomega coord deltat funcs
global h hh it NEN NN NGQ nx ny OEBC op ry soln tp tmax Umed xf

tp=0.0;
ArqEntr = fopen(strcat('ER_Progr2.inp'), 'r');
fgets(ArqEntr);
fgets(ArqEntr);
% Lendo demais parametros do problema
alpha      = fscanf(ArqEntr, 'alpha     :%f');     fgets(ArqEntr);
beta      = fscanf(ArqEntr, 'beta      :%f');     fgets(ArqEntr);
gamma      = fscanf(ArqEntr, 'gamma     :%f');     fgets(ArqEntr);
A1           = fscanf(ArqEntr, 'A1        :%f');     fgets(ArqEntr);
funcs.ws = fscanf(ArqEntr, 'wsFunc    :%s');     fgets(ArqEntr);
OEBC      = fscanf(ArqEntr, 'OEBC      :%d');     fgets(ArqEntr);
NEN       = fscanf(ArqEntr, 'NEN       :%d');     fgets(ArqEntr);
funcs.a1 = fscanf(ArqEntr, 'a1Func    :%s');    fgets(ArqEntr);
funcs.a2 = fscanf(ArqEntr, 'a2Func    :%s');    fgets(ArqEntr);
funcs.U  = fscanf(ArqEntr,  'UFunc    :%s');     fgets(ArqEntr);
funcs.VS = fscanf(ArqEntr,  'VSFunc   :%s');     fgets(ArqEntr);
op              = fscanf(ArqEntr,  'op        :%d');     fgets(ArqEntr);
funcs.b  = fscanf(ArqEntr, 'bFunc     :%s');     fgets(ArqEntr);
funcs.omega = fscanf(ArqEntr, 'omega :%s');     fgets(ArqEntr);
funcs.rho0  = fscanf(ArqEntr, 'rho0Func :%s'); fgets(ArqEntr);
funcs.u0 = fscanf(ArqEntr, 'u0Func       :%s'); fgets(ArqEntr);
NGQ             = fscanf(ArqEntr, 'NGQ       :%d');     fgets(ArqEntr);
% Número de pontos das coordenadas
nx            =  fscanf(ArqEntr, 'nx        :%d');     fgets(ArqEntr);
ny            =  fscanf(ArqEntr, 'ny        :%d');     fgets(ArqEntr);
% Dimensões do reservatório 2D
ax            =  fscanf(ArqEntr, 'ax        :%f');     fgets(ArqEntr);
ry            =  fscanf(ArqEntr, 'ry        :%f');     fgets(ArqEntr);
% fração do comprimento para profundidade atingir hmax
xf            =  fscanf(ArqEntr, 'xf        :%f');     fgets(ArqEntr);
% Valores das condições de contorno prescritas (EBC)
funcs.CE = fscanf(ArqEntr, 'CEFunc    :%s');     fgets(ArqEntr);
funcs.CD = fscanf(ArqEntr, 'CDFunc    :%s');     fgets(ArqEntr);
funcs.CInf = fscanf(ArqEntr, 'CInfFunc    :%s');fgets(ArqEntr);
funcs.CSup = fscanf(ArqEntr, 'CSupFunc    :%s');fgets(ArqEntr);
```

```
C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr242d2conc.m
Página 2 de 15                                                        28/09/2017 16:39:38
% Incremento de tempo e tempo máximo de cálculo
deltat   =  fscanf(ArqEntr, 'deltat   :%f');     fgets(ArqEntr);
tmax     =  fscanf(ArqEntr, 'tmax     :%f');     fgets(ArqEntr);
tol      = fscanf(ArqEntr, 'tol      :%f');    fgets(ArqEntr);
fclose(ArqEntr);
% Gera a Malha no início
gera_malha();
% Cálculo de parâmetros para aplicação das CC
hh = int32(ny);h=(coord(bpL(hh),2)-coord(bpL(1),2));  lth=ax/(nx-1);
A2 = - 0.5*(eval(funcs.ws)+A1)
Umax = A2+A1
Umed = 2*Umax/3
% Dimensiona matrizes e vetores
soln.Phi = zeros(NN,1); soln.Phi0 = zeros(NN,1); soln.Phi1 = zeros(NN,1);
soln.B = zeros(NN,NN); soln.C = zeros(NN,NN);
soln.CW=zeros (NN,1);
soln.K0 = zeros(NN,NN);
soln.M0 = zeros(NN,NN); soln.M1 = zeros(NN,NN); soln.MK = zeros(NN,NN);
CIomega = zeros(NN,1); soln.omega = zeros(NN,1); soln.omega2 = zeros(NN,1);
soln.omega3 = zeros(NN,1);
soln.u = zeros  (NN,1); soln.v = zeros  (NN,1);
% CIA = zeros(NN,1); soln.C1A = zeros(NN,1);
 CIB = zeros(NN,1); soln.C1B = zeros(NN,1);
% Define/calcula parâmetros do escoamento
Re = h*Umed*eval(funcs.rho0)/eval(funcs.u0)
Pe = Umed*h/(sqrt(eval(funcs.a1).^2+eval(funcs.a2).^2))
Da = eval(funcs.b)*h/Umed
Fr = (Umed.^2)/(9.81*h)
% Chama rotinas de resoluçao do sistema MEF
calcS();
calcSL();
calcSGlobal();
%Determina matrizes auxiliares
soln.M0=soln.M1; soln.K0 = soln.K; soln.MK=sparse(soln.M1\soln.K);
%soln.MDy=sparse(soln.M1\soln.Dy); %soln.MDx=sparse(soln.M1\soln.Dx);

%Preparação das condições de contorno da função-corrente e velocidade
for ij = 1:length(bpL)
     soln.MK(bpL(ij),:) = 0.0;   soln.MK(bpL(ij),bpL(ij)) = 1.0;
%      soln.MDx(bpL(ij),:) = 0.0; soln.MDx(bpL(ij),bpL(ij)) = 1.0;
%      soln.MDy(bpL(ij),:) = 0.0; soln.MDy(bpL(ij),bpL(ij)) = 1.0;
end
for ik=1:length(bpR)
    soln.MK(bpR(ik),:) = 0.0;   soln.MK(bpR(ik),bpR(ik)) = 1.0;
% Caso velocidade seja prescrita na saída
%   if OEBC == 1
%
%     soln.MDx(bpR(ik),:) = 0.0; soln.MDx(bpR(ik),bpR(ik)) = 1.0;
%     soln.MDy(bpR(ik),:) = 0.0; soln.MDy(bpR(ik),bpR(ik)) = 1.0;
%
%   elseif ik < int32((ny-1)/2-ny/5)|| ik > int32((ny-1)/2+ny/10)
%
%      soln.MDx(bpR(ik),:) = 0.0; soln.MDx(bpR(ik),bpR(ik)) = 1.0;
%      soln.MDy(bpR(ik),:) = 0.0; soln.MDy(bpR(ik),bpR(ik)) = 1.0;
%   end

```

```
end
for il=1:length(bpI)
     soln.MK(bpI(il),:) = 0.0;   soln.MK(bpI(il),bpI(il)) = 1.0;
%      soln.MDx(bpI(il),:) = 0.0; soln.MDx(bpI(il),bpI(il)) = 1.0;
%     soln.MDy(bpI(il),:) = 0.0; soln.MDy(bpI(il),bpI(il)) = 1.0;
end
for im = 1:length(bpS)
     soln.MK(bpS(im),:) = 0.0;    soln.MK(bpS(im),bpS(im)) = 1.0;
end
%
while le(tp,tmax)
% Aplica as condições iniciais
 if tp==0.0
%  if it == 1
    soln.omega(:,1) = eval(funcs.omega);
   for im = 1:length(bpS)
         soln.omega(bpS(im),1) = eval(funcs.ws);
   end
 else
  calcSB_C()
% Prepara matrizes para o transp das espécies
  soln.AC = (sparse(soln.MB)+(deltat*gamma)*sparse(soln.B-soln.C+soln.K3+ ...
              eval(funcs.b)*soln.MB));
  soln.BC = (sparse(soln.MB)-(deltat*(1-gamma))*sparse(soln.B-soln.C+soln.K3+ ...
              eval(funcs.b)*soln.MB));
% Esquema alternativo caso coeficientes de difusividade e de reação não sejam constantes
%  soln.AC = (sparse(soln.MB)+(deltat*gamma)*sparse(soln.B-soln.C+soln.K3+soln.MB2));
%  soln.BC = (sparse(soln.MB)-(deltat*(1-gamma))*sparse(soln.B-soln.C+soln.K3+soln.MB2));
%
% Localiza as condições de contorno para transp concentração
 aplCC2()
% Transporte da Concentração das Espécies (Esquema implícito)
% soln.C1A = (sparse(soln.M0)+(deltat/2)*sparse(soln.B-soln.C+soln.K2))\ ...
%  ((sparse(soln.M0)-(deltat/2)*sparse(soln.B-soln.C+soln.K2))*CIA);
% A fórmula acima só se aplica nos casos em que não se usar derivada material
% como CC de saída no cômputo das espécies.
  soln.C1B = sparse(soln.AC)\sparse(soln.BC)*CIB;
%Prepara as matrizes e vetor para o trnp da vorticidade
  if OEBC == 1||OEBC ==2
     soln.AW = (sparse(soln.M0)+(deltat*gamma)*sparse(soln.B-soln.C+ ...
                (eval(funcs.u0)/eval(funcs.rho0))*soln.K));
     soln.BW = (sparse(soln.M0)-(deltat*(1-gamma))*sparse(soln.B-soln.C+ ...
                (eval(funcs.u0)/eval(funcs.rho0))*soln.K));
     soln.CW =  - 9.81*deltat*beta*((1-gamma)*soln.Dx*CIB+ ...
               gamma*soln.Dx*soln.C1B)/eval(funcs.rho0);
  else
     soln.AW = (sparse(soln.MC)+(deltat*gamma)*sparse(soln.B-soln.C+ ...
                (eval(funcs.u0)/eval(funcs.rho0))*soln.K));
     soln.BW = (sparse(soln.MC)-(deltat*(1-gamma))*sparse(soln.B-soln.C+ ...
                (eval(funcs.u0)/eval(funcs.rho0))*soln.K));
     soln.CW =  - 9.81*deltat*beta*((1-gamma)*soln.DXC*CIB+ ...
               gamma*soln.DXC*soln.C1B)/eval(funcs.rho0);
```

C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr242d2conc.m
Página 4 de 15 **28/09/2017 16:39:38**

```
  end
% end
% Localiza as condições de contorno para transp vorticidade
  aplCC3()
% Transporte da vorticidade (Esquema implícito)
  soln.omega = sparse(soln.AW)\sparse(soln.BW)*CIomega+sparse(soln.CW);
 end
% Localiza as condições de contorno para determinação de Phi
  aplCC1()
% Determinação da Função-Corrente
  soln.Phi = soln.MK\soln.omega2;
% Localiza as CC para determinação do perfil de velocidades
%  aplCC1A()
% Determinação do perfil de velocidades
%  soln.u = sparse(soln.MDy)*soln.Phi0;
%  soln.v = -sparse(soln.MDx)*soln.Phi1;
% Localiza as condições de contorno da vorticidade com parâmetro de relaxação
  soln.omega3 = alpha*(diag(sum(soln.M0'))\(soln.K0*soln.Phi))+(1-alpha)*soln.omega;
  for im = 1:length(bpS)
        soln.omega3(bpS(im),1) = eval(funcs.U)*eval(funcs.ws);
%    if tp<=0.25*tmax
%        soln.omega3(bpS(im),1) = 0.0*eval(funcs.ws);
%    elseif tp<=0.5*tmax||tp>0.75*tmax
%        soln.omega3(bpS(im),1) = eval(funcs.U)*eval(funcs.ws);
%    elseif tp<=0.75*tmax
%        soln.omega3(bpS(im),1) = -eval(funcs.U)*eval(funcs.ws);
%    end
  end
% Pós-processamento
 CIomega = soln.omega;
% CIA = soln.C1A;
% CIA somente se usar CC de saída diferente da MDBC
 CIB = soln.C1B;
% imprime os resultados
 saidas()
%  Gera a Malha subsequente
% gera_malha();
% calcSGlobal();
 tp = tp + deltat
end
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
function aplCC1()
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
global A1 A2 bpL bpR bpI bpS coord funcs h  ny soln tp

soln.omega2=soln.omega;
 for ij = 1:length(bpL)
      x= (coord(bpL(ij),2)-coord(bpL(1),2))/h;
      soln.omega2(bpL(ij)) = eval(funcs.U)*(A1*(x.^2)/2+A2*(x.^3)/3);
 end
 for ik=1:length(bpR)
   if ik>=int32((ny-1)/2-ny/5)&&ik<=int32((ny-1)/2+ny/10)

        soln.omega2(bpR(ik)) = (eval(funcs.U)*(coord(bpR(ik),2)-coord(bpR(int32((ny-1)/2-ny/5)),2))) ...
```



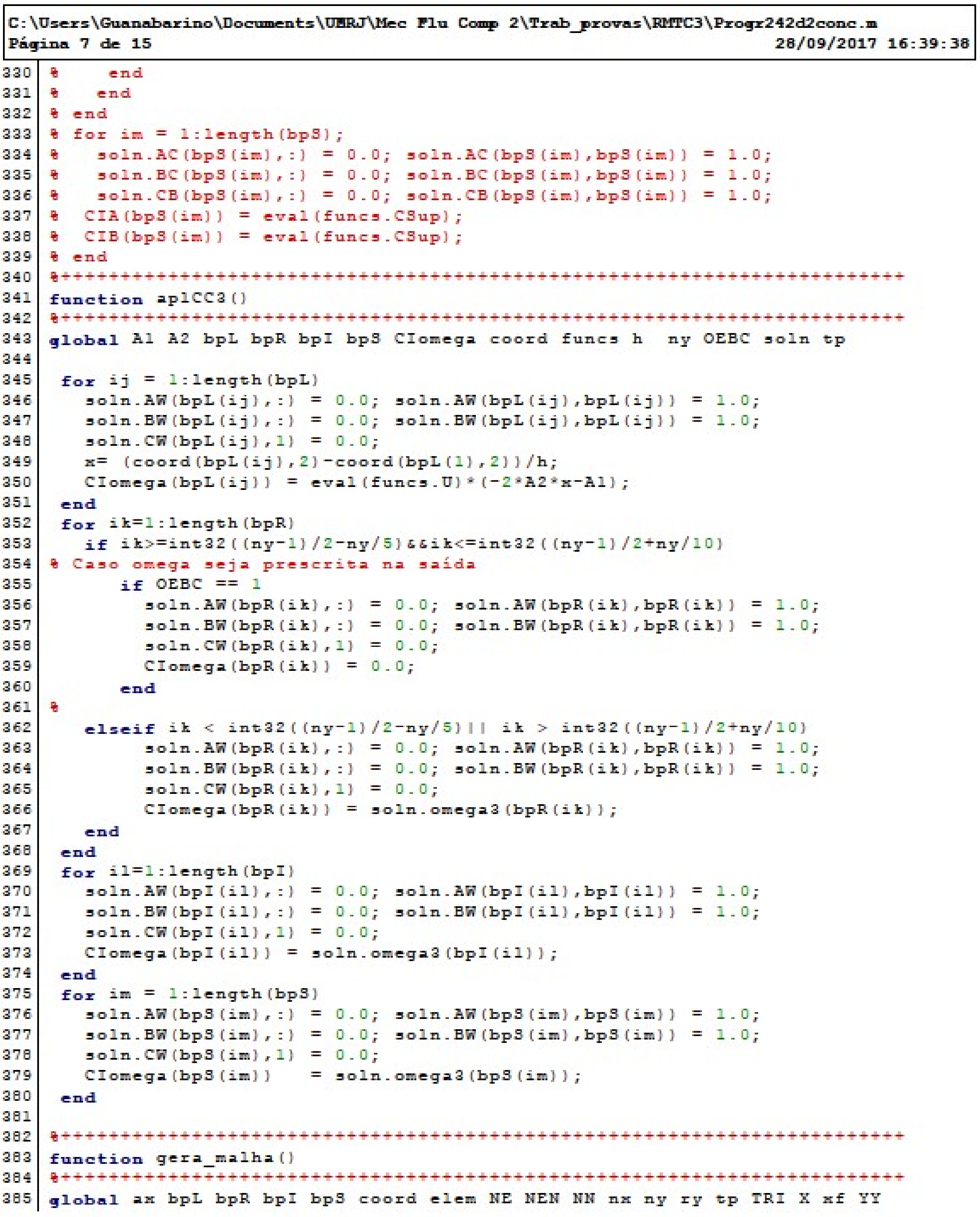
C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr242d2conc.m
Página 5 de 15   28/09/2017 16:39:38

```
        /(coord(bpR(int32((ny-1)/2+ny/10)),2)-coord(bpR(int32((ny-1)/2-ny/5)),2));
   elseif ik < int32((ny-1)/2-ny/5)
      soln.omega2(bpR(ik)) = eval(funcs.U)*0.0;
   elseif ik > int32((ny-1)/2+ny/10)
      soln.omega2(bpR(ik)) = eval(funcs.U)*1.0;
   end
 end
 for il=1:length(bpI)
   soln.omega2(bpI(il)) = eval(funcs.U)*0.0;
 end
 for im = 1:length(bpS)
   soln.omega2(bpS(im)) = eval(funcs.U)*1.0;
 end
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
function aplCC1A()
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
global A1 A2 bpL bpR bpI  coord funcs h  ny OEBC  soln tp

soln.Phi0=soln.Phi; % soln.Phi1=soln.Phi;
 for ij = 1:length(bpL)
      x= (coord(bpL(ij),2)-coord(bpL(1),2))/h;
      soln.Phi0(bpL(ij)) = eval(funcs.U)*(A1*x+A2*(x.^2));
%      soln.Phi1(bpL(ij)) = 0.0;
 end
 for ik=1:length(bpR)
%   soln.Phi1(bpR(ik)) = 0.0;
   if ik>=int32((ny-1)/2-ny/5)&&ik<=int32((ny-1)/2+ny/10)
% Caso a velocidade seja prescrita na saída
      if OEBC == 1
        soln.Phi0(bpR(ik)) = eval(funcs.VS);
      end
%
   else
        soln.Phi0(bpR(ik)) = 0.0;
   end
 end
 for il=1:length(bpI)
   soln.Phi0(bpI(il)) = 0.0;
%   soln.Phi1(bpI(il)) = 0.0;
 end
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
function aplCC2()
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
global bpL bpR bpI bpS CIA CIB deltat funcs  nx ny soln tmax tp xf

 for ij = 1:length(bpL)
   soln.AC(bpL(ij),:) = 0.0; soln.AC(bpL(ij),bpL(ij)) = 1.0;
   soln.BC(bpL(ij),:) = 0.0; soln.BC(bpL(ij),bpL(ij)) = 1.0;
%   if ij>=1 && ij<=hh;
    if tp>=deltat
%     CIA(bpL(ij))=eval(funcs.CE);
     CIB(bpL(ij))=eval(funcs.CE);
    else
     CIA(bpL(ij))=0.0;
     CIB(bpL(ij))=0.0;
```

- 5 -

```
C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr242d2conc.m
Página 6 de 15                                                         28/09/2017 16:39:38
```

```
   end
%  elseif ij > int32(((ny-1)/2)-ny/10)
%     CIA(bpL(ij))= 0.0
%     CIB(bpL(ij))=0.0;
%  end
end
for ik=1:length(bpR)
  soln.AC(bpR(ik),:) = 0.0; soln.AC(bpR(ik),bpR(ik)) = 1.0;
  soln.BC(bpR(ik),:) = 0.0; soln.BC(bpR(ik),bpR(ik)) = 1.0;
  if ik>=int32((ny-1)/2-ny/5)&&ik<=int32((ny-1)/2+ny/10)
% Caso a concentração de saída seja prescrita
%    CIA(bpR(ik)) = eval(funcs.CD);
%    CIB(bpR(ik)) = eval(funcs.CD);
  elseif ik < int32((ny-1)/2-ny/5)
%     CIA(bpR(ik))= eval(funcs.CD);
     CIB(bpR(ik))= eval(funcs.CD);
  elseif ik > int32((ny-1)/2+ny/10)
%     CIA(bpR(ik))=eval(funcs.CD);
     CIB(bpR(ik))=eval(funcs.CD);
  end
end
for il=1:length(bpI)
  soln.AC(bpI(il),:) = 0.0; soln.AC(bpI(il),bpI(il)) = 1.0;
  soln.BC(bpI(il),:) = 0.0; soln.BC(bpI(il),bpI(il)) = 1.0;
%  CIA(bpI(il)) = 0.0;
  CIB(bpI(il)) = 0.0;
end
  if tp<0.25*tmax
   for ij=1:length(bpI)
%    CIA(bpI(ij)) = eval(funcs.CInf);
    CIB(bpI(ij)) = eval(funcs.CInf);
   end
  end
  if tp>=0.25*tmax&&tp<0.4*tmax
   for ij=int32(length(bpI)/10):int32(2*length(bpI)/nx):length(bpI)
%    CIA(bpI(ij)) = eval(funcs.CInf);
    CIB(bpI(ij)) = eval(funcs.CInf);
   end
  end
  if tp>=0.4*tmax&&tp<=tmax
   for ij=int32(length(bpI)/8):int32(3*length(bpI)/nx):length(bpI)
%    CIA(bpI(ij)) = eval(funcs.CInf);
    CIB(bpI(ij)) = eval(funcs.CInf);
   end
  end
%  if tp>=0.6*tmax&&tp<0.75*tmax
%   for ij=int32(length(bpI)/5):int32(4*length(bpI)/nx):length(bpI)
%    CIA(bpI(ij)) = eval(funcs.CInf);
%    CIB(bpI(ij)) = eval(funcs.CInf);
%   end
%  end
%  if tp>=0.75*tmax&&tp<=tmax
%   for ij=int32(0.2*length(bpI)):int32(length(bpI)/8):length(bpI)
%   for ij=int32(length(bpI)/3):int32(5*length(bpI)/nx):length(bpI)
%    CIA(bpI(ij)) = eval(funcs.CInf);
%    CIB(bpI(ij)) = eval(funcs.CInf);
```



C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr242d2conc.m
Página 7 de 15 28/09/2017 16:39:38

```matlab
%     end
%    end
% end
% for im = 1:length(bpS);
%    soln.AC(bpS(im),:) = 0.0; soln.AC(bpS(im),bpS(im)) = 1.0;
%    soln.BC(bpS(im),:) = 0.0; soln.BC(bpS(im),bpS(im)) = 1.0;
%    soln.CB(bpS(im),:) = 0.0; soln.CB(bpS(im),bpS(im)) = 1.0;
%   CIA(bpS(im)) = eval(funcs.CSup);
%   CIB(bpS(im)) = eval(funcs.CSup);
% end
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
function aplCC3()
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
global A1 A2 bpL bpR bpI bpS CIomega coord funcs h  ny OEBC soln tp

 for ij = 1:length(bpL)
   soln.AW(bpL(ij),:) = 0.0; soln.AW(bpL(ij),bpL(ij)) = 1.0;
   soln.BW(bpL(ij),:) = 0.0; soln.BW(bpL(ij),bpL(ij)) = 1.0;
   soln.CW(bpL(ij),1) = 0.0;
   x= (coord(bpL(ij),2)-coord(bpL(1),2))/h;
   CIomega(bpL(ij)) = eval(funcs.U)*(-2*A2*x-A1);
 end
 for ik=1:length(bpR)
   if ik>=int32((ny-1)/2-ny/5)&&ik<=int32((ny-1)/2+ny/10)
% Caso omega seja prescrita na saída
      if OEBC == 1
        soln.AW(bpR(ik),:) = 0.0; soln.AW(bpR(ik),bpR(ik)) = 1.0;
        soln.BW(bpR(ik),:) = 0.0; soln.BW(bpR(ik),bpR(ik)) = 1.0;
        soln.CW(bpR(ik),1) = 0.0;
        CIomega(bpR(ik)) = 0.0;
      end
%
   elseif ik < int32((ny-1)/2-ny/5)|| ik > int32((ny-1)/2+ny/10)
        soln.AW(bpR(ik),:) = 0.0; soln.AW(bpR(ik),bpR(ik)) = 1.0;
        soln.BW(bpR(ik),:) = 0.0; soln.BW(bpR(ik),bpR(ik)) = 1.0;
        soln.CW(bpR(ik),1) = 0.0;
        CIomega(bpR(ik)) = soln.omega3(bpR(ik));
   end
 end
 for il=1:length(bpI)
   soln.AW(bpI(il),:) = 0.0; soln.AW(bpI(il),bpI(il)) = 1.0;
   soln.BW(bpI(il),:) = 0.0; soln.BW(bpI(il),bpI(il)) = 1.0;
   soln.CW(bpI(il),1) = 0.0;
   CIomega(bpI(il)) = soln.omega3(bpI(il));
 end
 for im = 1:length(bpS)
   soln.AW(bpS(im),:) = 0.0; soln.AW(bpS(im),bpS(im)) = 1.0;
   soln.BW(bpS(im),:) = 0.0; soln.BW(bpS(im),bpS(im)) = 1.0;
   soln.CW(bpS(im),1) = 0.0;
   CIomega(bpS(im))   = soln.omega3(bpS(im));
 end

%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
function gera_malha()
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
global ax bpL bpR bpI bpS coord elem NE NEN NN nx ny ry tp TRI X xf YY
```

```

%Gera o grid
NN=nx*ny
[X,Y] = meshgrid(0:ax/(nx-1):ax, 0:ry/(ny-1):ry);
X = reshape(X,1,[]); Y = reshape(Y,1,[]);
% Locais das condiçoes de contorno
Xmin=min(X);bpL=find(X==Xmin);
Xmax=max(X);bpR=find(X==Xmax);
Ymin=min(Y);bpI=find(Y==Ymin);
Ymax=max(Y);bpS=find(Y==Ymax);
%Preparar o trapesium
X = X/ax; Y = Y/ry;
xM=X>=xf; xm=X<xf;
fx=xm*0.5.*(1+cos(pi*X/xf));
YY = (1-(ry-1).*fx./ry).*Y+(ry-1).*fx./ry;
%YY=(Y.*(1+4*X)/5+4*(1-X)/5).*xm+Y.*xM;
%YY = YY+0.02*sin(2*pi*3.5*X+tp).*YY;
%Ler e armazenar as coordenadas
coord (:,1) = X(:).*ax; coord(:,2)= YY(:).*ry;
%coord (:,1) = X(:); coord(:,2)= YY(:);
if NEN==3
%Gera os elementos
 TRI=delaunay(X,Y);
 NE=size(TRI,1)
%Localiza os elementos
 for e=1:NE
  for j=1:3
     elem(e).LtoG(j) = TRI(e,j);
  end
 end
elseif NEN == 4
 NE=(nx-1)*(ny-1)
 fk=zeros(nx,ny);
 QUADR=zeros(NE,4); nel=0;
 for j=1:ny
  for i=1:nx
      fk(i,j)=j+(i-1)*ny;
  end
 end
 for j=1:(ny-1)
  for i=1:(nx-1)
      nel = nel+1;
      QUADR(nel,:)=[fk(i,j) fk(i+1,j) fk(i+1,j+1) fk(i,j+1)];
  end
 end
 for e=1:NE
  for j=1:4
      elem(e).LtoG(j) = QUADR(e,j);
  end
 end
end

%========================================================================
function calcS()
%========================================================================
global GQ NGQ NEN S dS
```

```

if NEN == 3
 if NGQ == 3   % Elemento Triangular, quadratura de 3 ptos.
   GQ.point(1,1) = 0.5;      GQ.point(1,2) = 0.0;
   GQ.point(2,1) = 0.0;      GQ.point(2,2) = 0.5;
   GQ.point(3,1) = 0.5;      GQ.point(3,2) = 0.5;
   GQ.weight(1) = 1/6;
   GQ.weight(2) = 1/6;
   GQ.weight(3) = 1/6;
 elseif NGQ == 4   % Elemento Triangular, quadratura de 4 ptos.
   GQ.point(1,1) = 1/3;      GQ.point(1,2) = 1/3;
   GQ.point(2,1) = 0.6;      GQ.point(2,2) = 0.2;
   GQ.point(3,1) = 0.2;      GQ.point(3,2) = 0.6;
   GQ.point(4,1) = 0.2;      GQ.point(4,2) = 0.2;
   GQ.weight(1) = -27/96;
   GQ.weight(2) = 25/96;
   GQ.weight(3) = 25/96;
   GQ.weight(4) = 25/96;
 elseif NGQ == 7    % Elemento Triangular, quadratura de 7 ptos.
   GQ.point(1,1) = 1/3;                  GQ.point(1,2) = 1/3;
   GQ.point(2,1) = 0.059715871789770;    GQ.point(2,2) = 0.470142064105115;
   GQ.point(3,1) = 0.470142064105115;    GQ.point(3,2) = 0.059715871789770;
   GQ.point(4,1) = 0.470142064105115;    GQ.point(4,2) = 0.470142064105115;
   GQ.point(5,1) = 0.101286507323456;    GQ.point(5,2) = 0.797426985353087;
   GQ.point(6,1) = 0.101286507323456;    GQ.point(6,2) = 0.101286507323456;
   GQ.point(7,1) = 0.797426985353087;    GQ.point(7,2) = 0.101286507323456;
   GQ.weight(1) = 0.225 / 2;
   GQ.weight(2) = 0.132394152788 / 2;
   GQ.weight(3) = 0.132394152788 / 2;
   GQ.weight(4) = 0.132394152788 / 2;
   GQ.weight(5) = 0.125939180544 / 2;
   GQ.weight(6) = 0.125939180544 / 2;
   GQ.weight(7) = 0.125939180544 / 2;
 end
 for k = 1:NGQ
    ksi = GQ.point(k,1); eta = GQ.point(k,2);
    S(1,k) = 1 - ksi - eta;
    S(2,k) = ksi;
    S(3,k) = eta;
% derivadas de S em ksi
    dS(1,1,k) = -1;
    dS(1,2,k) =  1;
    dS(1,3,k) =  0;
% derivadas de S em eta
    dS(2,1,k) = -1;
    dS(2,2,k) =  0;
    dS(2,3,k) =  1;
 end
elseif NEN == 4
 if NGQ == 4   % Elemento Quadrilateral quadratura de 4 ptos.
    GQ.point(1,1) = -sqrt(1/3);   GQ.point(1,2) = -sqrt(1/3);
    GQ.point(2,1) = sqrt(1/3);    GQ.point(2,2) = -sqrt(1/3);
    GQ.point(3,1) = -sqrt(1/3);   GQ.point(3,2) = sqrt(1/3);
    GQ.point(4,1) = sqrt(1/3);    GQ.point(4,2) = sqrt(1/3);
    GQ.weight(1) = 1.0;
    GQ.weight(2) = 1.0;
    GQ.weight(3)  = 1.0;
    GQ.weight(4)  = 1.0;
 elseif NGQ == 9   % Elemento Quadrilateral quadratura de 9 ptos.
    GQ.point(1,1) = -sqrt(3/5);    GQ.point(1,2) = -sqrt(3/5);
    GQ.point(2,1) = 0.0;           GQ.point(2,2) = -sqrt(3/5);
    GQ.point(3,1) = sqrt(3/5);     GQ.point(3,2) = -sqrt(3/5);
    GQ.point(4,1) = -sqrt(3/5);    GQ.point(4,2) = 0.0;
    GQ.point(5,1) = 0.0;           GQ.point(5,2) = 0.0;
    GQ.point(6,1) = sqrt(3/5);     GQ.point(6,2) = 0.0;
    GQ.point(7,1) = -sqrt(3/5);    GQ.point(7,2) = sqrt(3/5);
    GQ.point(8,1) = 0.0;           GQ.point(8,2) = sqrt(3/5);
    GQ.point(9,1) = sqrt(3/5);     GQ.point(9,2) = sqrt(3/5);
    GQ.weight(1)  = 5/9 * 5/9;
    GQ.weight(2)  = 8/9 * 5/9;
    GQ.weight(3)  = 5/9 * 5/9;
    GQ.weight(4)  = 5/9 * 8/9;
    GQ.weight(5)  = 8/9 * 8/9;
    GQ.weight(6)  = 5/9 * 8/9;
    GQ.weight(7)  = 5/9 * 5/9;
    GQ.weight(8)  = 8/9 * 5/9;
    GQ.weight(9)  = 5/9 * 5/9;
 end
 for k = 1:NGQ
     ksi = GQ.point(k,1); eta = GQ.point(k,2);
     S(1,k)  = 0.25*(1-ksi)*(1-eta);
     S(2,k)  = 0.25*(1+ksi)*(1-eta);
     S(3,k)  = 0.25*(1+ksi)*(1+eta);
     S(4,k)  = 0.25*(1-ksi)*(1+eta);
  % derivadas de S em ksi
      dS(1,1,k)  = -0.25*(1-eta);
      dS(1,2,k)  =  0.25*(1-eta);
      dS(1,3,k)  =  0.25*(1+eta);
      dS(1,4,k)  = -0.25*(1+eta);
  % derivadas de S em eta
      dS(2,1,k)  = -0.25*(1-ksi);
      dS(2,2,k)  = -0.25*(1+ksi);
      dS(2,3,k)  =  0.25*(1+ksi);
      dS(2,4,k)  =  0.25*(1-ksi);
 end
end
%=========================================================================
function calcSL()
%=========================================================================
global dSL NEN SL GQL

SL=zeros(NEN, NEN);
% Quadratura de três pontos
GQL.point(1)  = -sqrt(3/5);
GQL.point(2)  = 0;
GQL.point(3)  = sqrt(3/5);
GQL.weight(1) = 5/9;
GQL.weight(2) = 8/9;
GQL.weight(3) = 5/9;
% Funções de forma Quadráticas de Lagrange
% Valores da função de forma para uso na CC de saída
for k = 1:3
```

```
    ksi = GQL.point(k);
    SL(1,k)   = 0.5 * ksi * (ksi - 1);
    SL(2,k)   = 1 - ksi * ksi;
    SL(3,k)   = 0.5 * ksi * (1 + ksi);
    dSL(1,k)   = -0.5 + ksi;
    dSL(2,k)   = -2 * ksi;
    dSL(3,k)   = 0.5 + ksi;
end
%=====================================================================
function calcSGlobal()
%=====================================================================
global  NE NEN NN NGQ soln coord dS elem bpR

JL=zeros(length(bpR));
% Calculo do Jacobiano e seu determinante para cada elemento 2-D.
% inicializa matriz de coordenadas
e_coord=zeros(NEN,2);
for e = 1:NE
    elem(e).JL=0.0;
end
for e = 1:NE
 for i = 1:NEN
    iG = elem(e).LtoG(i);
    e_coord(i,:) = coord(iG,:);
% calculo do jacobiano da integral de linha para a CC direita derivada materia
l
   for kk=1:length(bpR)
    if iG == bpR(kk)
     if kk < length(bpR)
       dx = coord(bpR(kk),1)-coord(bpR(kk+1),1);
       dy = coord(bpR(kk),2)-coord(bpR(kk+1),2);
       JL(kk)= sqrt(dx*dx+dy*dy)/2;
       elem(e).JL=JL(kk);
     end
    end
   end
 end
% Calculo e arma. das derivadas das funções de forma.
 for k = 1:NGQ
    Jacob(:,:) = dS(:,:,k) * e_coord(:,:);
    elem(e).gDS(:,:,k) = (Jacob(:,:)) \ dS(:,:,k);
    elem(e).detJacob(k) = det(Jacob);
 end
end
% Calculo das matrizes [D], [K]s, [M]s
 soln.Dx = zeros(NN,NN); soln.DxL = zeros(NN,NN);soln.Dy = zeros(NN,NN);
 soln.K = zeros(NN,NN);  % soln.K1 = zeros(NN,NN);
% soln.K2 = zeros(NN,NN);
 soln.K3 = zeros(NN,NN);
 soln.MB = zeros(NN,NN);
 %soln.MB2 = zeros(NN,NN);
 soln.MC = zeros(NN,NN);
 soln.M11 = zeros(NN,NN);
 soln.M2 = zeros(NN,NN);
 %soln.M21 = zeros(NN,NN);
 soln.M22 = zeros(NN,NN);
```

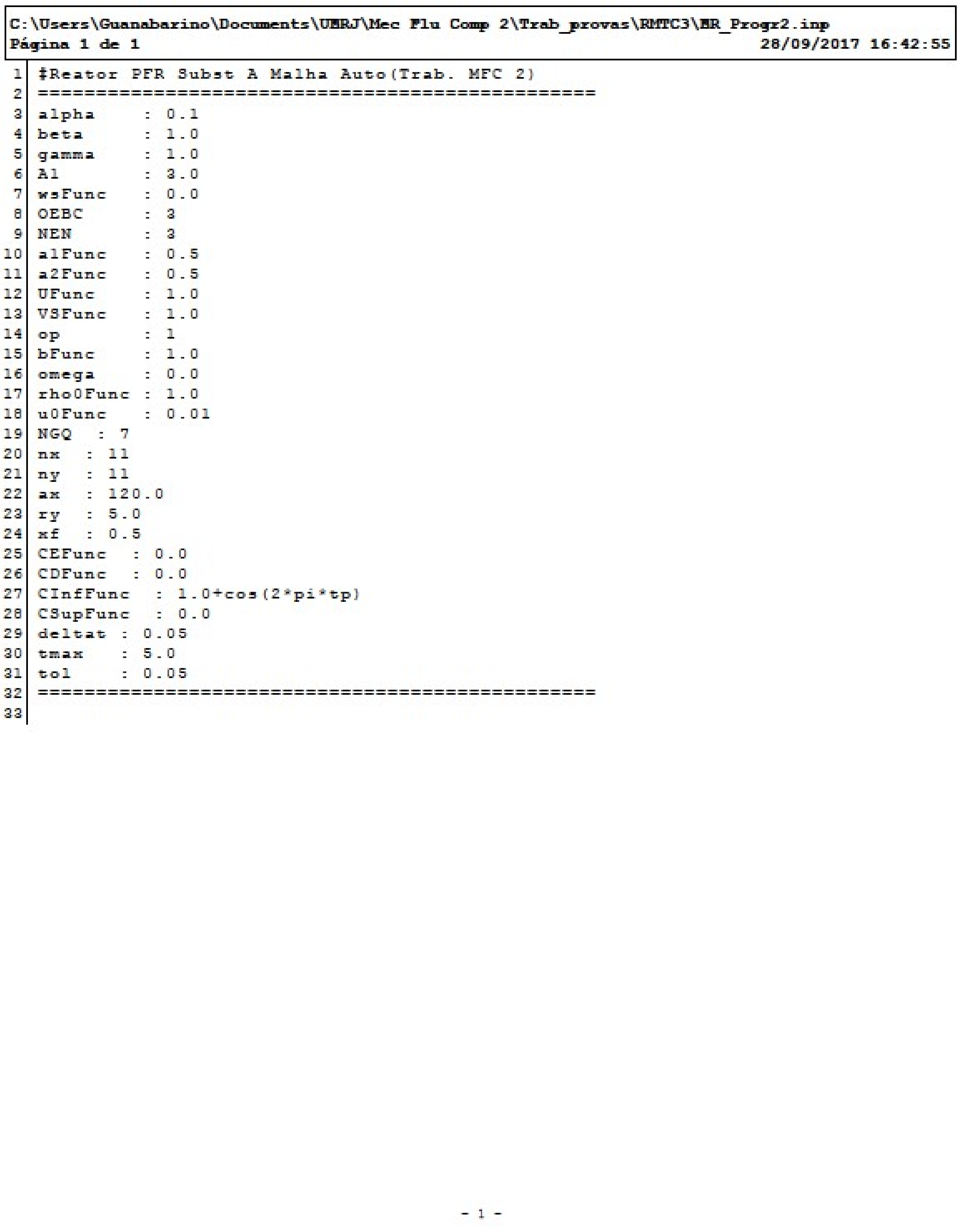
```
for e = 1:NE
  calcSElem(e);
  monta(e);
end
 soln.MB=soln.M1+soln.M2;  %soln.MB2=soln.M11+soln.M21;
 soln.MC=soln.M1+soln.M22; soln.DXC = soln.Dx+soln.DxL;
%=======================================================================
function calcSElem(e)
%=======================================================================
% Calculo de [K]'s' e [M]'s' no elemento
global  bValue coord dSL elem funcs OEBC GQ GQL NEN NGQ S SL Umed tp

elem(e).Dxe = zeros(NEN,NEN); elem(e).Dye = zeros(NEN,NEN);
elem(e).DxLe = zeros(NEN,NEN);
elem(e).Ke = zeros(NEN,NEN); % elem(e).K1e = zeros(NEN,NEN);
%elem(e).K2e = zeros(NEN,NEN);
elem(e).K3e = zeros(NEN,NEN);
elem(e).M1e = zeros(NEN,NEN);% elem(e).M11e = zeros(NEN,NEN);
elem(e).M2e = zeros(NEN,NEN);% elem(e).M21e = zeros(NEN,NEN);
elem(e).M22e = zeros(NEN,NEN);

for k = 1:NGQ
 x = 0;
 y = 0;
 for i = 1:NEN
    iG = elem(e).LtoG(i);
    x = x + S(i,k) * coord(iG,1);
    y = y + S(i,k) * coord(iG,2);
 end

 a1Value = eval(funcs.a1);
 a2Value = eval(funcs.a2);
% bValue = eval(funcs.b)
 rho0Value = eval(funcs.rho0);
 u0Value = eval(funcs.u0);

 for i = 1:NEN
  for j = 1:NEN

    elem(e).Dxe(i,j)  = elem(e).Dxe(i,j)  + S(i,k)*elem(e).gDS(1,j,k)  ...
                        * elem(e).detJacob(k)* GQ.weight(k);
    elem(e).Dye(i,j)  = elem(e).Dye(i,j)  + S(i,k)*elem(e).gDS(2,j,k)  ...
                        * elem(e).detJacob(k)* GQ.weight(k);
    elem(e).Ke(i,j)  = elem(e).Ke(i,j)  + (elem(e).gDS(1,i,k)  ...
                        * elem(e).gDS(1,j,k)+ elem(e).gDS(2,i,k)  ...
                        * elem(e).gDS(2,j,k))  ...
                        * elem(e).detJacob(k)* GQ.weight(k);
%    elem(e).K1e(i,j)  =     elem(e).K1e(i,j)+(u0Value/rho0Value)*elem(e).Ke(i,
j);
%     elem(e).K2e(i,j)  = elem(e).K2e(i,j)  +  (a1Value*elem(e).gDS(1,i,k)  ...
%                         * elem(e).gDS(1,j,k)+ a2Value*elem(e).gDS(2,i,k)  ...
%                         * elem(e).gDS(2,j,k)+ bValue * S(i,k)  * S(j,k))  ...
%                         * elem(e).detJacob(k)* GQ.weight(k);
     elem(e).K3e(i,j)  = elem(e).K3e(i,j)  +  (a1Value*elem(e).gDS(1,i,k)  ...
                        * elem(e).gDS(1,j,k)+ a2Value*elem(e).gDS(2,i,k)  ...
                        * elem(e).gDS(2,j,k))*elem(e).detJacob(k)* GQ.weight(k
```

C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr242d2conc.m
Página 13 de 15 28/09/2017 16:39:38

```
;
      elem(e).Mle(i,j) =  elem(e).Mle(i,j) + S(i,k) * S(j,k)*elem(e).detJacob(k
)* GQ.weight(k);
%      elem(e).M1le(i,j) = elem(e).M1le(i,j)+bValue*elem(e).Mle(i,j);
   end
  end
end

if OEBC == 3
 for k = 1:3
  x = 0;
  y = 0;
  for i = 1:3
    iG = elem(e).LtoG(i);
    x = x + SL(i,k) * coord(iG,1);
    y = y + SL(i,k) * coord(iG,2);
  end

  a1Value = eval(funcs.a1);
  a2Value = eval(funcs.a2);
  aValue=sqrt(a1Value*a1Value+a2Value*a2Value);
%  bValue = eval(funcs.b)
  rho0Value = eval(funcs.rho0);
  u0Value = eval(funcs.u0);
  UValue = Umed*eval(funcs.U);

  for i = 1:NEN
   for j = 1:NEN
    elem(e).M2e(i,j)= elem(e).M2e(i,j)+(aValue/UValue)*SL(i,k) ...
    *SL(j,k)*elem(e).JL*GQL.weight(k);
%    elem(e).M21e(i,j)= elem(e).M21e(i,j)+bValue*elem(e).M2e(i,j);
    elem(e).M22e(i,j)= elem(e).M22e(i,j)+(u0Value/(rho0Value*UValue)) ...
    *SL(i,k)*SL(j,k)*elem(e).JL*GQL.weight(k);
    elem(e).DxLe(i,j)= elem(e).DxLe(i,j)+(u0Value/(rho0Value*UValue)) ...
    *SL(i,k)*dSL(j,k)*GQL.weight(k);
   end
  end
  end
end
%=========================================================================
function monta(e)
%=========================================================================
global elem NEN soln

% Monta De em D, Ke's' em K's' e Me's' em M's'

for i = 1:NEN
  iG = elem(e).LtoG(i);
 for j = 1:NEN
    jG = elem(e).LtoG(j);
    soln.Dx(iG,jG) = soln.Dx(iG,jG) + elem(e).Dxe(i,j);
    soln.DxL(iG,jG) = soln.DxL(iG,jG) + elem(e).DxLe(i,j);
    soln.Dy(iG,jG) = soln.Dy(iG,jG) + elem(e).Dye(i,j);
    soln.K(iG,jG) = soln.K(iG,jG) + elem(e).Ke(i,j);
%    soln.K1(iG,jG) = soln.K1(iG,jG) + elem(e).K1e(i,j);
%    soln.K2(iG,jG) = soln.K2(iG,jG) + elem(e).K2e(i,j);
```

```
     soln.K3(iG,jG) = soln.K3(iG,jG) + elem(e).K3e(i,j);
     soln.M1(iG,jG) = soln.M1(iG,jG) + elem(e).M1e(i,j);
     soln.M2(iG,jG) = soln.M2(iG,jG) + elem(e).M2e(i,j);
     soln.M22(iG,jG) = soln.M22(iG,jG) + elem(e).M22e(i,j);
  end
end
%=======================================================================
function calcSB_C()
%=======================================================================
global elem GQ NE NEN NGQ  S  soln

for e = 1:NE
 elem(e).Be = zeros(NEN,NEN); elem(e).Ce = zeros(NEN,NEN);
 for k = 1:NGQ
 dpsidy=0; dpsidx=0;
  for m=1:NEN
      dpsidy=dpsidy + elem(e).gDS(2,m,k) * soln.Phi(elem(e).LtoG(m));
      dpsidx=dpsidx + elem(e).gDS(1,m,k) * soln.Phi(elem(e).LtoG(m));
  end
  for i = 1:NEN
    for j = 1:NEN
        elem(e).Be(i,j) = elem(e).Be(i,j) + S(i,k) *dpsidy * ...
                elem(e).gDS(1,j,k) * elem(e).detJacob(k) * GQ.weight(k);
        elem(e).Ce(i,j) = elem(e).Ce(i,j) + S(i,k) * dpsidx * ...
                elem(e).gDS(2,j,k)* elem(e).detJacob(k) * GQ.weight(k);

    end
  end
 end

 for i = 1:NEN
   iG = elem(e).LtoG(i);
  for j = 1:NEN
    jG = elem(e).LtoG(j);
    soln.B(iG,jG) = soln.B(iG,jG) + elem(e).Be(i,j);
    soln.C(iG,jG) = soln.C(iG,jG) + elem(e).Ce(i,j);
  end
 end
end
%=======================================================================
function saidas()
%=======================================================================
global ax ry  NE NEN NGQ  op coord soln elem


%Grafico dos resultados
x=zeros(NEN,NE);y=zeros(NEN,NE);
% z=zeros(NEN,NE);t=zeros(NEN,NE);
Phi=zeros(NEN,NE);Omega=zeros(NEN,NE);t=zeros(NEN,NE);vel=zeros(NEN,NE);
% C=X*0; xx=coord(:,1); yy=coord(:,2);
for e = 1:NE
 for i = 1:NEN
    x(i,e) = coord(elem(e).LtoG(i),1);
    y(i,e) = coord(elem(e).LtoG(i),2);
%     z(i,e) = soln.C1A(elem(e).LtoG(i));
%     TRI(e,i)=elem(e).LtoG(i);
%     Phi(i,e)  = soln.Phi(elem(e).LtoG(i));
%     Omega(i,e)= soln.omega(elem(e).LtoG(i));
%     C(TRI(e,i))=soln.C1A(elem(e).LtoG(i));
     t(i,e)  = soln.C1B(elem(e).LtoG(i));
%     vel(i,e)  = soln.u(elem(e).LtoG(i));
 end
end
% Plotagem das figuras
figure(1)
%hold on;
%subplot(2,2,1)
%patch(x,y,Phi); axis([0 ax 0 1.05*ry]); colorbar
%xlabel('x'); ylabel('y'); zlabel('C');
%title(sprintf('Stream Function'));
%subplot(2,2,2)
%patch(x,y,Omega); axis([0 ax 0 1.05*ry]); colorbar
%xlabel('x'); ylabel('y'); zlabel('C');
%title(sprintf('Vorticity'));
% figure(2)
 subplot(2,1,1)
if op==1
 patch(x,y,t); axis([0 ax 0 1.05*ry]);
% caxis([0.0 2.0]);
% az = 0; el = 90; view(az, el);
 xlabel('x'); ylabel('y'); zlabel('C');
else
 patch(x,y,vel); axis([0 ax 0 1.05*ry]);
% caxis([0.0 2.0]);
% az = 0; el = 90; view(az, el);
 xlabel('x'); ylabel('y'); zlabel('u ou v');
end
if op==1
 if NEN==3
  title(sprintf('Concentration Profile \nTriangular Elements - Gauss Quadrature %d points', NGQ));
 elseif NEN==4
  title(sprintf('Concentration Profile \nQuadrangular Elements - Gauss Quadrature %d points', NGQ));
 end
else
 if NEN==3
  title(sprintf('Longitudinal Velocity Profile \nTriangular Elements - Gauss Quadrature %d points', NGQ));
 elseif NEN==4
  title(sprintf('Longitudinal Velocity Profile \nQuadrangular Elements - Gauss Quadrature %d points', NGQ));
 end
end
hold off
```

B.2 Code inputs sample file



```
C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\ER_Progr2.inp
Página 1 de 1                                                    28/09/2017 16:42:55
```

```
#Reator PFR Subst A Malha Auto(Trab. MFC 2)
=================================================
alpha     : 0.1
beta      : 1.0
gamma     : 1.0
A1        : 3.0
wsFunc    : 0.0
OEBC      : 3
NEN       : 3
a1Func    : 0.5
a2Func    : 0.5
UFunc     : 1.0
VSFunc    : 1.0
op        : 1
bFunc     : 1.0
omega     : 0.0
rho0Func : 1.0
u0Func    : 0.01
NGQ  : 7
nx   : 11
ny   : 11
ax   : 120.0
ry   : 5.0
xf   : 0.5
CEFunc  : 0.0
CDFunc  : 0.0
CInfFunc  : 1.0+cos(2*pi*tp)
CSupFunc  : 0.0
deltat : 0.05
tmax   : 5.0
tol    : 0.05
=================================================

```

- 1 -

APPENDIX C – LID-DRIVEN CAVITY TEST MATLAB CODE

C.1 Source code

```
C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr22d12b.m
Página 1 de 9                                                    28/09/2017 17:13:45
```

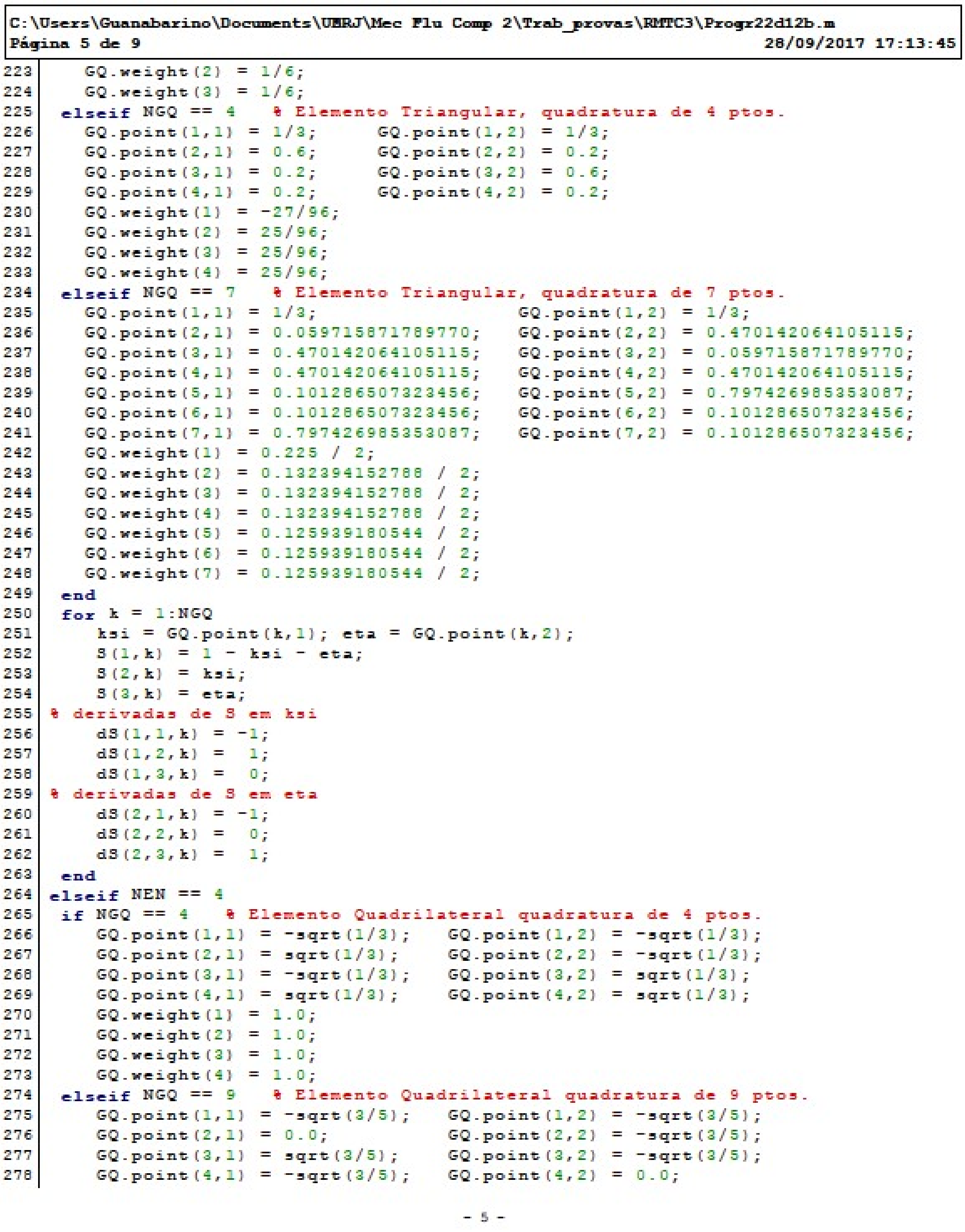
```
function [] = Progr22d12b
%=========================================================================
% Teste do código em cavidade
%=========================================================================
clc; clear all; close all;

global alpha ax beta bpL bpR bpI bpS CIomega coord deltat funcs
global h NEN NN NGQ nx ny op Re   ry soln tp tmax xf

tp=0.0;
ArqEntr = fopen(strcat('ER_Progr2cav.inp'), 'r');
fgets(ArqEntr);
fgets(ArqEntr);
% Lendo demais parametros do problema
% Obs: O arquivo tem vários parâmatros que não serão utilizados,
% mas foi mantido assim para aproveitar o mesmo do código de escoamento
alpha     = fscanf(ArqEntr, 'alpha      :%f');   fgets(ArqEntr);
beta     = fscanf(ArqEntr, 'beta       :%f');   fgets(ArqEntr);
gamma     = fscanf(ArqEntr, 'gamma      :%f');   fgets(ArqEntr);
NEN      = fscanf(ArqEntr, 'NEN      :%d');    fgets(ArqEntr);
funcs.a1 = fscanf(ArqEntr, 'a1Func     :%s');   fgets(ArqEntr);
funcs.a2 = fscanf(ArqEntr, 'a2Func     :%s');   fgets(ArqEntr);
funcs.U =  fscanf(ArqEntr,  'UFunc    :%s');    fgets(ArqEntr);
op          =  fscanf(ArqEntr,  'op       :%d');    fgets(ArqEntr);
funcs.b  = fscanf(ArqEntr, 'bFunc     :%s');    fgets(ArqEntr);
funcs.omega = fscanf(ArqEntr, 'omega :%s');    fgets(ArqEntr);
funcs.rho0  = fscanf(ArqEntr, 'rho0Func :%s'); fgets(ArqEntr);
funcs.u0 = fscanf(ArqEntr, 'u0Func       :%s'); fgets(ArqEntr);
NGQ        = fscanf(ArqEntr, 'NGQ       :%d');    fgets(ArqEntr);
% Número de pontos das coordenadas
nx          =  fscanf(ArqEntr, 'nx       :%d');    fgets(ArqEntr);
ny          =  fscanf(ArqEntr, 'ny       :%d');    fgets(ArqEntr);
% Dimensões do reservatório 2D
ax          =  fscanf(ArqEntr, 'ax       :%f');    fgets(ArqEntr);
ry          =  fscanf(ArqEntr, 'ry       :%f');    fgets(ArqEntr);
% fração do comprimento para profundidade atingir hmax
xf          =  fscanf(ArqEntr, 'xf       :%f');    fgets(ArqEntr);
% Valores das condições de contorno prescritas (EBC)
funcs.CE = fscanf(ArqEntr, 'CEFunc    :%s');    fgets(ArqEntr);
funcs.CD = fscanf(ArqEntr, 'CDFunc    :%s');    fgets(ArqEntr);
funcs.CInf = fscanf(ArqEntr, 'CInfFunc    :%s');fgets(ArqEntr);
funcs.CSup = fscanf(ArqEntr, 'CSupFunc    :%s');fgets(ArqEntr);
deltat  =  fscanf(ArqEntr, 'deltat  :%f');     fgets(ArqEntr);
% tempo máximo de cálculo
tmax     =  fscanf(ArqEntr, 'tmax     :%f');     fgets(ArqEntr);
tol       = fscanf(ArqEntr, 'tol     :%f');    fgets(ArqEntr);
fclose(ArqEntr);
% Gera a Malha no início
gera_malha();
% Parâmetro para avaliação de padrões de escoamento
hh = int32(ny); h=(coord(bpL(hh),2)-coord(bpL(1),2))/coord(bpL(hh),2);
% Dimensiona matrizes e vetores
soln.Phi = zeros(NN,1);
soln.B = zeros(NN,NN); soln.C = zeros(NN,NN);
soln.K = zeros(NN,NN);
soln.K1 = zeros(NN,NN);
```

C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr22d12b.m
Página 2 de 9 **28/09/2017 17:13:45**

```
soln.M1 = zeros(NN,NN);
soln.BV = zeros(NN,1);
CIomega = zeros(NN,1); soln.omega = zeros(NN,1); soln.omega2 = zeros(NN,1);
soln.omega3 = zeros(NN,1);
% Define/calcula parâmetros do escoamento
 Re = abs(h*eval(funcs.U)*eval(funcs.rho0)/eval(funcs.u0));
% Chama rotinas de resoluçao do sistema MEF
calcS();
calcSGlobal();
%Determina matrizes auxiliares
soln.M0=soln.M1; soln.K0 = soln.K;
%Preparação das condições de contorno
for ij = 1:length(bpL)
     soln.M1(bpL(ij),:) = 0.0; soln.M1(bpL(ij),bpL(ij)) = 1.0;
     soln.K(bpL(ij),:) = 0.0;  soln.K(bpL(ij),bpL(ij)) = 1.0;
end
for ik=1:length(bpR)
     soln.M1(bpR(ik),:) = 0.0; soln.M1(bpR(ik),bpR(ik)) = 1.0;
     soln.K(bpR(ik),:) = 0.0;  soln.K(bpR(ik),bpR(ik)) = 1.0;
end
for il=1:length(bpI)
    soln.M1(bpI(il),:) = 0.0; soln.M1(bpI(il),bpI(il)) = 1.0;
    soln.K(bpI(il),:) = 0.0;  soln.K(bpI(il),bpI(il)) = 1.0;
end
for im = 1:length(bpS)
    soln.M1(bpS(im),:) = 0.0;  soln.M1(bpS(im),bpS(im)) = 1.0;
    soln.K(bpS(im),:) = 0.0;   soln.K(bpS(im),bpS(im)) = 1.0;
    soln.BV(bpS(im)) = eval(funcs.U)/(nx-1);
end
    soln.BV(bpS(1)) = 0.5*eval(funcs.U)/(nx-1);
    soln.BV(bpS(nx))= 0.5*eval(funcs.U)/(nx-1);
while tp<=tmax
% Aplica as condições iniciais
  if tp==0.0
       soln.omega(:,1) = eval(funcs.omega);
  elseif tp>0.0
  calcSB_C()
%Prepara as matrizes e vetor para o trnp da vorticidade
  soln.AW = ((soln.M0)+(deltat*gamma)*(soln.B-soln.C+soln.K1));
  soln.BW = ((soln.M0)-(deltat*(1-gamma))*(soln.B-soln.C+soln.K1));
% Localiza as condições de contorno para transp vorticidade
  aplCC3()
% Transporte da vorticidade
  soln.omega = sparse(soln.AW)\(sparse(soln.BW))*CIomega;
  end
% Localiza as condições de contorno para determinação de Phi(n)
  aplCC1()
% Determinação da Função-Corrente
  soln.Phi = soln.K\(soln.M1*soln.omega2+soln.BV);
% Localiza as condições de contorno da vorticidade com parâmetro de relaxação (ou não)
   soln.omega3 = alpha*(soln.K0*soln.Phi-soln.BV)./(sum(soln.M0)')+ (1-alpha)*soln.omega3;
% Pós-processamento
 CIomega = soln.omega;
 minPhi=min(soln.Phi)
```

```
C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr22d12b.m
Página 3 de 9                                                  28/09/2017 17:13:45
```

```
 Pmin = find(minPhi==soln.Phi)
 XP = coord(Pmin,1)
 YP = coord(Pmin,2)
 VP= soln.omega(Pmin)
% imprime os resultados
 saidas()
%incremento do tempo
 tp = tp + deltat
end
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
function aplCC1()
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
global bpL bpR bpI bpS  soln

soln.omega2=soln.omega;
 for ij = 1:length(bpL)
      soln.omega2(bpL(ij)) = 0.0;
 end
 for ik=1:length(bpR)
      soln.omega2(bpR(ik)) =  0.0;
 end
 for il=1:length(bpI)
   soln.omega2(bpI(il)) = 0.0;
 end
 for im = 1:length(bpS)
   soln.omega2(bpS(im)) = -soln.BV(bpS(im));
 end
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
function aplCC3()
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
global bpL bpR bpI bpS CIomega     soln


 for ij = 1:length(bpL)
   soln.AW(bpL(ij),:) = 0.0; soln.AW(bpL(ij),bpL(ij)) = 1.0;
   soln.BW(bpL(ij),:) = 0.0; soln.BW(bpL(ij),bpL(ij)) = 1.0;
  CIomega(bpL(ij))=soln.omega3(bpL(ij));
 end
 for ik=1:length(bpR)
   soln.AW(bpR(ik),:) = 0.0; soln.AW(bpR(ik),bpR(ik)) = 1.0;
   soln.BW(bpR(ik),:) = 0.0; soln.BW(bpR(ik),bpR(ik)) = 1.0;
    CIomega(bpR(ik))=soln.omega3(bpR(ik));
 end
 for il=1:length(bpI)
   soln.AW(bpI(il),:) = 0.0; soln.AW(bpI(il),bpI(il)) = 1.0;
   soln.BW(bpI(il),:) = 0.0; soln.BW(bpI(il),bpI(il)) = 1.0;
   CIomega(bpI(il))=soln.omega3(bpI(il));
 end
 for im = 1:length(bpS)
   soln.AW(bpS(im),:) = 0.0; soln.AW(bpS(im),bpS(im)) = 1.0;
   soln.BW(bpS(im),:) = 0.0; soln.BW(bpS(im),bpS(im)) = 1.0;
   CIomega(bpS(im))=soln.omega3(bpS(im));
 end
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
function gera_malha()
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
global  bpL bpR bpI bpS coord elem NE NEN NN nx ny  TRI X
global  Xmax Xmin Ymax Ymin

%Gera o grid
NN=nx*ny;
[X,Y] = meshgrid(0:1.0/(nx-1):1.0, 0:1.0/(ny-1):1.0);
X = reshape(X,1,[]); Y = reshape(Y,1,[]);
% Locais das condiçoes de contorno
Xmin=min(X);bpL=find(X==Xmin);
Xmax=max(X);bpR=find(X==Xmax);
Ymin=min(Y);bpI=find(Y==Ymin);
Ymax=max(Y);bpS=find(Y==Ymax);
%Ler e armazenar as coordenadas
coord (:,1) = X(:); coord(:,2)= Y(:);
if NEN==3
%Gera os elementos
 TRI=delaunay(X,Y);
 NE=size(TRI,1);
%Localiza os elementos
 for e=1:NE
  for j=1:3
     elem(e).LtoG(j) = TRI(e,j);
  end
 end
elseif NEN == 4
 NE=(nx-1)*(ny-1);
 fk=zeros(nx,ny);
 QUADR=zeros(NE,4); nel=0;
 for j=1:ny
  for i=1:nx
      fk(i,j)=j+(i-1)*ny;
  end
 end
 for j=1:(ny-1)
  for i=1:(nx-1)
      nel = nel+1;
      QUADR(nel,:)=[fk(i,j) fk(i+1,j) fk(i+1,j+1) fk(i,j+1)];
  end
 end
 for e=1:NE
  for j=1:4
      elem(e).LtoG(j) = QUADR(e,j);
  end
 end
end
%=========================================================================
function calcS()
%=========================================================================
global GQ NGQ NEN S dS

if NEN == 3
 if NGQ == 3   % Elemento Triangular, quadratura de 3 ptos.
   GQ.point(1,1) = 0.5;      GQ.point(1,2) = 0.0;
   GQ.point(2,1) = 0.0;      GQ.point(2,2) = 0.5;
   GQ.point(3,1) = 0.5;      GQ.point(3,2) = 0.5;
   GQ.weight(1) = 1/6;
```



C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr22d12b.m
Página 5 de 9 **28/09/2017 17:13:45**

```
   GQ.weight(2) = 1/6;
   GQ.weight(3) = 1/6;
 elseif NGQ == 4   % Elemento Triangular, quadratura de 4 ptos.
   GQ.point(1,1) = 1/3;     GQ.point(1,2) = 1/3;
   GQ.point(2,1) = 0.6;     GQ.point(2,2) = 0.2;
   GQ.point(3,1) = 0.2;     GQ.point(3,2) = 0.6;
   GQ.point(4,1) = 0.2;     GQ.point(4,2) = 0.2;
   GQ.weight(1) = -27/96;
   GQ.weight(2) = 25/96;
   GQ.weight(3) = 25/96;
   GQ.weight(4) = 25/96;
 elseif NGQ == 7   % Elemento Triangular, quadratura de 7 ptos.
   GQ.point(1,1) = 1/3;                GQ.point(1,2) = 1/3;
   GQ.point(2,1) = 0.059715871789770;   GQ.point(2,2) = 0.470142064105115;
   GQ.point(3,1) = 0.470142064105115;   GQ.point(3,2) = 0.059715871789770;
   GQ.point(4,1) = 0.470142064105115;   GQ.point(4,2) = 0.470142064105115;
   GQ.point(5,1) = 0.101286507323456;   GQ.point(5,2) = 0.797426985353087;
   GQ.point(6,1) = 0.101286507323456;   GQ.point(6,2) = 0.101286507323456;
   GQ.point(7,1) = 0.797426985353087;   GQ.point(7,2) = 0.101286507323456;
   GQ.weight(1) = 0.225 / 2;
   GQ.weight(2) = 0.132394152788 / 2;
   GQ.weight(3) = 0.132394152788 / 2;
   GQ.weight(4) = 0.132394152788 / 2;
   GQ.weight(5) = 0.125939180544 / 2;
   GQ.weight(6) = 0.125939180544 / 2;
   GQ.weight(7) = 0.125939180544 / 2;
 end
 for k = 1:NGQ
    ksi = GQ.point(k,1); eta = GQ.point(k,2);
    S(1,k) = 1 - ksi - eta;
    S(2,k) = ksi;
    S(3,k) = eta;
% derivadas de S em ksi
    dS(1,1,k) = -1;
    dS(1,2,k) =  1;
    dS(1,3,k) =  0;
% derivadas de S em eta
    dS(2,1,k) = -1;
    dS(2,2,k) =  0;
    dS(2,3,k) =  1;
 end
elseif NEN == 4
 if NGQ == 4   % Elemento Quadrilateral quadratura de 4 ptos.
    GQ.point(1,1) = -sqrt(1/3);   GQ.point(1,2) = -sqrt(1/3);
    GQ.point(2,1) = sqrt(1/3);    GQ.point(2,2) = -sqrt(1/3);
    GQ.point(3,1) = -sqrt(1/3);   GQ.point(3,2) = sqrt(1/3);
    GQ.point(4,1) = sqrt(1/3);    GQ.point(4,2) = sqrt(1/3);
    GQ.weight(1) = 1.0;
    GQ.weight(2) = 1.0;
    GQ.weight(3) = 1.0;
    GQ.weight(4) = 1.0;
 elseif NGQ == 9   % Elemento Quadrilateral quadratura de 9 ptos.
    GQ.point(1,1) = -sqrt(3/5);   GQ.point(1,2) = -sqrt(3/5);
    GQ.point(2,1) = 0.0;          GQ.point(2,2) = -sqrt(3/5);
    GQ.point(3,1) = sqrt(3/5);    GQ.point(3,2) = -sqrt(3/5);
    GQ.point(4,1) = -sqrt(3/5);   GQ.point(4,2) = 0.0;
```

- 5 -

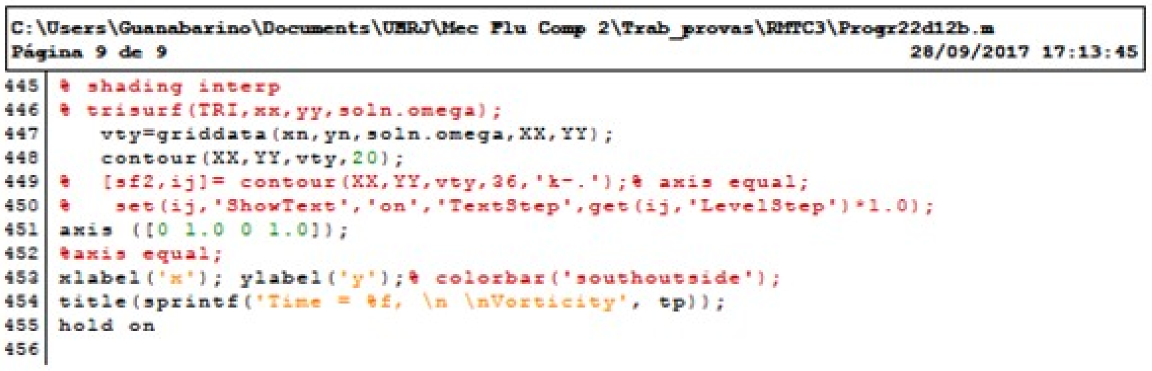
```
C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr22d12b.m
Página 6 de 9                                                      28/09/2017 17:13:45
    GQ.point(5,1) = 0.0;          GQ.point(5,2) = 0.0;
    GQ.point(6,1) = sqrt(3/5);    GQ.point(6,2) = 0.0;
    GQ.point(7,1) = -sqrt(3/5);   GQ.point(7,2) = sqrt(3/5);
    GQ.point(8,1) = 0.0;          GQ.point(8,2) = sqrt(3/5);
    GQ.point(9,1) = sqrt(3/5);    GQ.point(9,2) = sqrt(3/5);
    GQ.weight(1) = 5/9 * 5/9;
    GQ.weight(2) = 8/9 * 5/9;
    GQ.weight(3) = 5/9 * 5/9;
    GQ.weight(4) = 5/9 * 8/9;
    GQ.weight(5) = 8/9 * 8/9;
    GQ.weight(6) = 5/9 * 8/9;
    GQ.weight(7) = 5/9 * 5/9;
    GQ.weight(8) = 8/9 * 5/9;
    GQ.weight(9) = 5/9 * 5/9;
 end
 for k = 1:NGQ
     ksi = GQ.point(k,1); eta = GQ.point(k,2);
     S(1,k) = 0.25*(1-ksi)*(1-eta);
     S(2,k) = 0.25*(1+ksi)*(1-eta);
     S(3,k) = 0.25*(1+ksi)*(1+eta);
     S(4,k) = 0.25*(1-ksi)*(1+eta);
  % derivadas de S em ksi
      dS(1,1,k) = -0.25*(1-eta);
      dS(1,2,k) =  0.25*(1-eta);
      dS(1,3,k) =  0.25*(1+eta);
      dS(1,4,k) = -0.25*(1+eta);
  % derivadas de S em eta
      dS(2,1,k) = -0.25*(1-ksi);
      dS(2,2,k) = -0.25*(1+ksi);
      dS(2,3,k) =  0.25*(1+ksi);
      dS(2,4,k) =  0.25*(1-ksi);
 end
end
%========================================================================
function calcSGlobal()
%========================================================================
global  NE NEN  NGQ  coord dS elem Jacob

% Calculo do Jacobiano e seu determinante para cada elemento 2-D.
% inicializa matriz de coordenadas
e_coord=zeros(NEN,2);
for e = 1:NE
 for i = 1:NEN
    iG = elem(e).LtoG(i);
    e_coord(i,:) = coord(iG,:);
 end
% Calculo e arma. das derivadas das funções de forma.
 for k = 1:NGQ
    Jacob(:,:) = dS(:,:,k) * e_coord(:,:);
    elem(e).gDS(:,:,k) = (Jacob(:,:)) \ dS(:,:,k);
    elem(e).detJacob(k) = det(Jacob);
 end
end
% Calculo das matrizes [K] e [M]
for e = 1:NE
  calcSElem(e);
```

C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr22d12b.m
Página 7 de 9 28/09/2017 17:13:45

```
  monta(e);
end
%==========================================================================
function calcSElem(e)
%==========================================================================
% Calculo de [K],[M]'s' no elemento
global  elem GQ  NEN NGQ Re S

elem(e).Ke = zeros(NEN,NEN);  elem(e).K1e = zeros(NEN,NEN);
elem(e).M1e = zeros(NEN,NEN);

for k = 1:NGQ
 for i = 1:NEN
  for j = 1:NEN

    elem(e).Ke(i,j) = elem(e).Ke(i,j) + (elem(e).gDS(1,i,k) ...
                        * elem(e).gDS(1,j,k)+ elem(e).gDS(2,i,k) ...
                        * elem(e).gDS(2,j,k)) ...
                        * elem(e).detJacob(k)* GQ.weight(k);
     elem(e).K1e(i,j) =  elem(e).K1e(i,j)+elem(e).Ke(i,j)/Re;
     elem(e).M1e(i,j) =  elem(e).M1e(i,j) + S(i,k) * S(j,k)*elem(e).detJacob(k
)* GQ.weight(k);
  end
 end
end

%==========================================================================
function monta(e)
%==========================================================================
global elem NEN  soln

% Monta Ke em K, Me em M

for i = 1:NEN
  iG = elem(e).LtoG(i);
 for j = 1:NEN
    jG = elem(e).LtoG(j);
    soln.K(iG,jG) = soln.K(iG,jG) + elem(e).Ke(i,j);
    soln.K1(iG,jG) = soln.K1(iG,jG) + elem(e).K1e(i,j);
    soln.M1(iG,jG) = soln.M1(iG,jG) + elem(e).M1e(i,j);
 end
end
%==========================================================================
function calcSB_C()
%==========================================================================
global elem funcs GQ Jacob NE NEN NGQ nx S soln coord Xmax Xmin Ymax Ymin

for e = 1:NE
 elem(e).Be = zeros(NEN,NEN); elem(e).Ce = zeros(NEN,NEN);
 for k = 1:NGQ
 dpsidy=0; dpsidx=0;
  for m=1:NEN
        dpsidy=dpsidy + elem(e).gDS(2,m,k)* soln.Phi(elem(e).LtoG(m));
        dpsidx=dpsidx + elem(e).gDS(1,m,k)* soln.Phi(elem(e).LtoG(m));
  end
  for i = 1:NEN
```

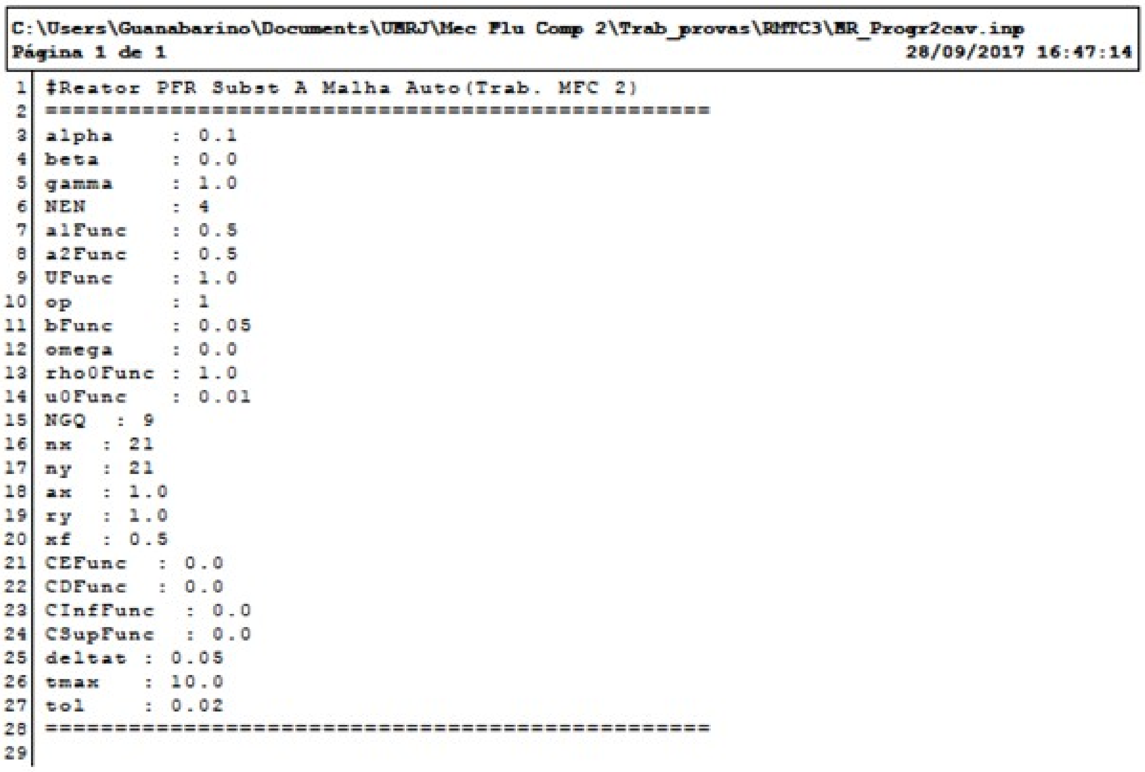
C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr22d12b.m
Página 8 de 9 28/09/2017 17:13:45

```
    for j = 1:NEN
        elem(e).Be(i,j) = elem(e).Be(i,j) + S(i,k)*dpsidy* ...
                elem(e).gDS(1,j,k) * elem(e).detJacob(k) * GQ.weight(k);
        elem(e).Ce(i,j) = elem(e).Ce(i,j) + S(i,k)*dpsidx* ...
                elem(e).gDS(2,j,k)* elem(e).detJacob(k) * GQ.weight(k);

    end
  end
 end
 for i = 1:NEN
   iG = elem(e).LtoG(i);
  for j = 1:NEN
    jG = elem(e).LtoG(j);
    soln.B(iG,jG) = soln.B(iG,jG) + elem(e).Be(i,j);
    soln.C(iG,jG) = soln.C(iG,jG) + elem(e).Ce(i,j);
  end
 end
end
%=========================================================================
function saidas()
%=========================================================================
global   CIomega tp NE NEN coord nx ny soln elem Re

%Grafico dos resultados
x=zeros(NEN,NE);y=zeros(NEN,NE);
xn=coord(:,1);yn=coord(:,2);
for e = 1:NE
 for i = 1:NEN
    x(i,e) = coord(elem(e).LtoG(i),1);
    y(i,e) = coord(elem(e).LtoG(i),2);
%    TRI(e,i)=elem(e).LtoG(i);
%    Phi(i,e) = soln.Phi(elem(e).LtoG(i));
%    Omega(i,e)= soln.omega(elem(e).LtoG(i));

 end
end
% Plotagem das figuras
figure(1)
hold off
subplot(1,2,1)
% patch(x,y,Phi); axis([0 1.0 0 1.0]);
% shading interp
% surf(x,y,Phi); axis([0 ax 0 1.05*ry]);
   xx=0:1.0/(nx-1):1.0; yy=0:1.0/(ny-1):1.0;
  [XX,YY]=meshgrid(xx,yy);
   sfn=griddata(xn,yn,soln.Phi,XX,YY);
%   contour(XX,YY,sfn,20);
  [~,h]= contour(XX,YY,sfn,'k-'); % axis equal;
   set(h,'ShowText','on','TextStep',get(h,'LevelStep')*1.0);
axis ([0 1.0 0 1.0]);
%axis equal;
xlabel('x'); ylabel('y');% colorbar('southoutside');
title(sprintf('Elements Mesh %d, NRe = %d \n  \nStream Function', NE, Re));
%figure(2)
subplot(1,2,2)
% patch(x,y,Omega); axis([0 1.0 0 1.0]);
```



```
C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\Progr22d12b.m
Página 9 de 9                                                    28/09/2017 17:13:45
% shading interp
% trisurf(TRI,xx,yy,soln.omega);
   vty=griddata(xn,yn,soln.omega,XX,YY);
   contour(XX,YY,vty,20);
%   [sf2,ij]= contour(XX,YY,vty,36,'k-.');% axis equal;
%    set(ij,'ShowText','on','TextStep',get(ij,'LevelStep')*1.0);
axis ([0 1.0 0 1.0]);
%axis equal;
xlabel('x'); ylabel('y');% colorbar('southoutside');
title(sprintf('Time = %f, \n \nVorticity', tp));
hold on
```

C.2 Code inputs sample file



```
C:\Users\Guanabarino\Documents\UERJ\Mec Flu Comp 2\Trab_provas\RMTC3\ER_Progr2cav.inp
Página 1 de 1                                                    28/09/2017 16:47:14
#Reator PFR Subst A Malha Auto(Trab. MFC 2)
==================================================
alpha     : 0.1
beta      : 0.0
gamma     : 1.0
NEN       : 4
a1Func    : 0.5
a2Func    : 0.5
UFunc     : 1.0
op        : 1
bFunc     : 0.05
omega     : 0.0
rho0Func  : 1.0
u0Func    : 0.01
NGQ   : 9
nx   : 21
ny   : 21
ax   : 1.0
ry   : 1.0
xf   : 0.5
CEFunc   : 0.0
CDFunc   : 0.0
CInfFunc   : 0.0
CSupFunc   : 0.0
deltat  : 0.05
tmax    : 10.0
tol     : 0.02
==================================================
```

ANNEX A – PAPER PRESENTED IN THE XXVI CONGRESSO NACIONAL DE MATEMÁTICA APLICADA E COMPUTACIONAL (CNMAC)

$$a_0 + a_1\alpha_R + a_2\left(\alpha_R^2 - \alpha_I^2\right) = 0 \text{ and, therefore: } \alpha_I = \pm\sqrt{\alpha_R^2 + \frac{a_1}{a_2}\alpha_R + \frac{a_0}{a_2}} \tag{8}$$

which, after substitution in Eq.(7), implies in:

$$\beta_I = \pm\sqrt{\alpha_R^2 + \frac{a_1}{a_2}\alpha_R + \frac{a_0}{a_2}}\,(a_1 + 2a_2\alpha_R) \tag{9}$$

By noting that the concentration c cannot take negative values, in order to provide a physically consistent solution, we need to add a constant forcing such that this nonnegative restriction is satisfied. Also, for a constant in time forcing, we have $\beta_R = \beta_I = 0$ and, therefore, substituting this condition in Eq.(4), we obtain:

$$\hat{\alpha} = -\frac{a_1}{2a_2} \pm \sqrt{\frac{a_1^2}{4a_2^2} - \frac{a_0}{a_2}} \tag{10}$$

As expected, Eq.(2) has negative real solutions in case of $a_0 < 0$, thus producing a steady solution that decays in the x direction, representing the amplitude decay with the distance. So, given a_0, a_1, a_2, and an arbitrary α_R, we may construct a solution employing Eq.(3) and Eqs.(8) to (10).

2.2 Sample code test

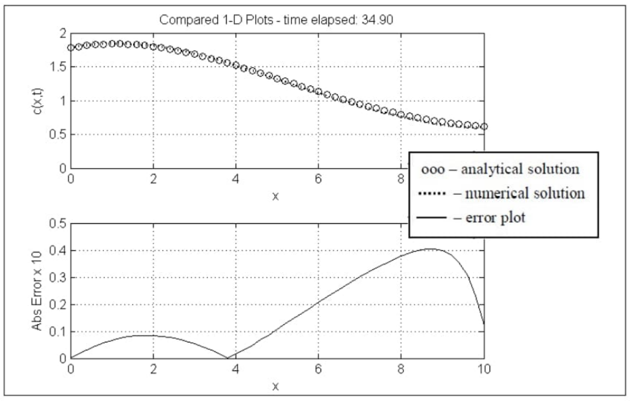


Figure 1: 1D Sample Code Graphics Output.

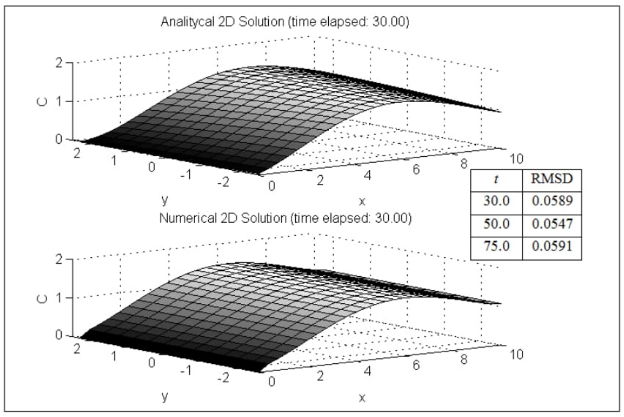
We show a test (Fig. 1) where a simple 1D explicit finite difference (FD) code that solves Eq.(2) is tested against the analytical solution given above. Entry parameters are: L (computational domain size) = 10; $\Delta x = 0.2$; $\Delta t = \Delta x^2/2$; $a_0 = -0.01$; $a_1 = -1.0$; $a_2 = 1.0$ and α_R is arbitrarily set as -0.1. The transient analytical and numerical solutions are plotted on the upper graph and the absolute error (amplified by 10) is plotted below.

Also, to assure the nonnegative restriction for the concentration profile, we must have also $\beta_I = 0$ and this supplies the constant forcing:

$$c_0 = e^{(\alpha_{xR}+i\alpha_{xI})x+i\alpha_{yI}y+i\alpha_{zI}z} \tag{21}$$

For the 2D case, we can proceed analogously, considering that there is not the z component and, as a consequence, periodicity in the z direction. Thus, given a_0, a_1, a_2 and arbitraries α_{xR}, α_{yI} and α_{zI}, according to the case, we may construct a 2D or 3D solution, employing Eqs.(18) to (21).

3.2 Sample code test



t	RMSD
30.0	0.0589
50.0	0.0547
75.0	0.0591

Figure 2: 2D Sample Code Graphics Output (1250 elements mesh).

A 2D code employing a simple Galerkin formulation through finite elements method (FEM) is employed for test and the outcome for a given elapsed time is shown by Fig. 2. The entry parameters are L = 10; W = 5; $\Delta x = 0.4$; $\Delta y = 0.4$; $\Delta t = 0.1$; $a_0 = -0.01$; $a_{1x} = -2.0$; $a_{1y} = -0.2$; $a_2 = 1.0$; α_{xR} and α_{yI} are arbitrarily set as -0.1. The error is evaluated through Root Mean Square Deviation (RMSD), which is also shown in Fig. 2, or:

$$\text{RMSD} = \sqrt{\frac{\sum_{i=1}^{m}(C_i - C_i^a)^2}{m}} \tag{22}$$

where C_i^a is the analytical solution at node i for a given total number of nodes m.

We further observe that for 2D, Eq.(21) assumes the form:

$$c_0 = e^{\hat{\alpha}_x x + i\alpha_{yI}y} \tag{23}$$

ANNEX B – PAPER TO BE PUBLISHED IN THE BRAZILIAN JOURNAL OF CHEMICAL ENGINEERING (BJChE)

SIMULATION OF SPECIES CONCENTRATION DISTRIBUTION IN REACTIVE FLOWS WITH UNSTEADY BOUNDARY CONDITIONS

A. G. de Oliveira Filho[1], N. Mangiavacchi[2] and J. Pontes[3]

UERJ – Universidade do Estado do Rio de Janeiro, GESAR
Rua Fonseca Teles 121, São Cristóvão, 20940-903
Phone: +(55)(21) 2332-4733, Rio de Janeiro – RJ, Brazil
Email: [1]ade_oliveira@hotmail.com, [2]norberto @uerj.br, [3]jose.pontes@uerj.br

Abstract – The determination of species concentration profiles in reactive flows with variable inlets is a problem of practical interest to many fields such as in flow reactor transient operation and in cyclic degradable pollutants disposals in watercourses. In these cases, the inflow condition often consists of a time-dependent function which may imply in unsteady outflows, not always well represented by the usual boundary conditions (BC) so far used. A new approach, using an outlet condition in the form of a material derivative, termed Material Derivative Boundary Condition (MDBC), is introduced and a numerical model to solve convection-diffusion-reaction equations in two-dimensional (2-D) incompressible flows is developed. Upon reviewing the literature, it is noticed that Finite Element Method (FEM) is rarely used in the simulation of reactive flows, in spite of its ability of consistently coping with variable BCs. The above facts are reasons to explore its use along with a semi-discrete formulation with Galerkin Method in our simulations. Results are obtained for various conditions, in order to show features of the code and are compared to existing solutions. Use of MDBC is shown to provide a better approximation of the exit concentrations and use of FEM in reactive flows is further enhanced.

Keywords — Concentration Profile Simulation, 2-D Reactive Flows, Finite Element Method, Material Derivative, Unsteady Boundary Conditions.

INTRODUCTION

Preliminaries

The determination of species concentration profiles in incompressible reactive flows presents practical interest to many engineering applications, such as tubular continuous chemical reactors design and operation, concentration evolution prediction of degradable and non-buoyant contaminants in rivers, downstream industrial wastewater or domestic sewage discharge, etc.

While reactants in chemical reactors are subject to transformation due to chemical or biochemical reactions, pollutants in rivers may also disappear by physical processes, such as volatilization or reactive decay, all of which being accounted for in the transport equation by addition of a reaction term *r* (van der Perk, 2013):

$$\frac{\partial C}{\partial t} = -\overline{u}_i \frac{\partial C}{\partial x_i} + \frac{\partial}{\partial x_i}\left[D_{ij} \frac{\partial C}{\partial x_j} \right] \pm r \qquad (1)$$

where we define for 2-D flows: $D_{ij} = \begin{bmatrix} D_x & 0 \\ 0 & D_y \end{bmatrix}$.

After a certain initial time interval, when the mixing processes are completed, species concentration along the flow can be modeled by the use of equation 1. In ideal tube reactors, often treated as plug flow devices, molecular diffusion and radial/lateral velocities terms may be dropped (Levenspiel, 1999), leading to one-dimensional (1-D) pure advective-reactive model. In other cases, these terms must be taken into account, requiring 2-D models to describe the flow. It is also reasonable to assume 1-D convective and diffusive flows for small rivers and channels when the length is ten or more times larger than its width (Kachiashvili et al., 2007). In larger watercourses, by its turn, where the river depth is significantly small compared to its width, depth-averaged concentrations assuming vertically well-mixed species could be employed (Lee and Seo, 2007), making it possible to apply a 2-D model derived from equation 1.

Thus, it is all about solving equation 1 in the applicable dimensions, subject to proper initial and boundary conditions Usually, three types of BC apply:

$$C = \bar{c} \qquad \text{on} \quad \Gamma_e \tag{2}$$

$$\frac{\partial C}{\partial x_i} = \bar{q} \qquad \text{on} \quad \Gamma_n \tag{3}$$

$$\bar{u} C + \bar{D} \frac{\partial C}{\partial x_i} = \bar{g} \qquad \text{on} \quad \Gamma_r \tag{4}$$

where $\bar{c}$, $\bar{q}$ and $\bar{g}$ may be homogeneous, constant valued or function of time and the greek letters Γ denote the correspondent surface where the BC applies. Equation 2 is usually referred to as Dirichlet or Essential Boundary Condition (EBC), equation 3, as Neumann or Natural Boundary Condition (NBC) and equation 4, as Robin or Cauchy Boundary Condition.

Scope

In this paper, we are particularly interested in 2-D simulations of reacting species transport, where the inlet boundary concentration is a pulse, a series of pulses or a continuous periodic function. These inlet conditions apply to cases of flow chemical reactors operating under variable inlet feed and variable species concentration spills in rivers and channels.

This class of problems has motivated studies pursuing analytical solutions of convection-diffusion-reaction equations–subjected to time-dependent–BCs, like the ones from van Genuchten and Alves (1982), Logan and Zlotnik (1995), Logan (1996), Aral and Liao (1996), Golz and Dorroh (2001), Chen and Liu (2011) and Pérez Guerrero et al. (2013). However, these studies either are restricted to 1-D cases, or adopt conditions that may not represent time dependence close to the domain exit.

We emphasize that, in the case of time-dependent inlet conditions, special attention must be given to the outlet BC. Since the exit concentration or the species flux is an unknown, assuming prescribed values at the outlet is not consistent.

To the present, as in the works cited above, this indeterminacy is treated either by considering that the outlet concentration gradients are zero, which may be physically unrealistic (Ziskind et al., 2011), or by using Robin type BCs, best suited to represent inlet conditions.

Literature Review

A number of papers address the advection–dispersion equation, with or without the reaction term, providing both analytical and numerical solutions for cases of pollutants discharge. O'Loughlin and Bowmer (1975), for instance, applied analytical solutions to equation 1 in 1-D channel flows with decaying species, later extended by Chapman (1979) to non-uniform steady rivers, both considering only pulse or continuous inlet concentrations and homogeneous NBC for the concentrations at the outlet. Comparison with the results obtained in the experimental works of Vilhena and Leal (1981) for non-reacting pollutants in point source injection shows good agreement with them. Czernuszenko (1987), also working with dispersion of conservative species, proposed a numerical solution for the 2-D advection–diffusion equation, using a conditionally stable finite differences (FD) scheme. But, since the study was restricted to mixing far from the pollution source, leaving convection to the background, the equation was bounded by NBCs, not encompassing unsteady BCs. Piasecki and Katopodes (1997) interested in sensitivity of contaminant concentration profiles to timely changes in its load, a similar aspect of our own concern, treated the problem by the use of a FEM scheme, but the unsteady load was a zeroth order production term of the transport equation and the problem was subjected to Dirichlet and Neumann type BCs. Kaschiashvili et al. (2007) provided a consistent model for river reactive flow problems in one, two and three dimensions and used dimension-splitting FD numerical schemes, with unsteady upstream BC and a NBC downstream. But, due to the equilibrium condition at the outlet, consisting of a constant spatial concentration gradient, this BC no longer applies and is modified, sometimes, with the introduction of an additional parameter in order to better reproduce experimental data. The fact supports our remark that time-dependent inlet conditions may imply in difficulties for prescribing values for the outlet conditions. Lee and Seo (2007) used a 2-D finite element model, based on the Streamline-Upwind Petrov-Galerkin Method (SUPG) together with a Crank-Nicholson FD scheme for the time derivative, as in this paper, but restricted to rivers where the process is diffusion dominated and the downstream BC was a prescribed diffusion flux. Two years later, the same authors employed this same method to accidental mass release in rivers (Lee and Seo, 2009) and, similarly to Piasecki and Katopodes (1997), the accidental mass release was represented by a zeroth order production term of the transport equation which was subjected to Dirichlet inlet BC and Neumann outlet BC, once more not considering unsteady BCs.

The literature survey detailed above, related to watercourses pollutants spills, shows that FEM has not been widely used to obtain solutions of reactive flows, in spite of its ability of consistently coping with differential BCs (Logan, 2007). This might be explained by the existence of the advective term in the transport equation that makes the system of equations nonsymmetric and prone to numerical oscillations (Yu and Singh, 1995). Several authors addressed the problem by focusing the development of consistent and stable FEM schemes for these flows (Yu and Singh, 1995; Galeão et al., 2004; John and Schmeyer, 2011) but rarely holding their attention on unsteady BCs. We also quote the studies of Konzen et al (2007) by which a convective-diffusive-reactive problem formulated through vorticity and stream-function is numerically solved, employing Galerkin FEM (GFEM) together with a Runge-Kutta scheme for the time stepping. But, owing to the formulation adopted, the BCs were assumed homogeneous Neumann type and the flow, taking place in a closed cavity, is not subjected to inflows and outflows rates, as in rivers and continuous chemical reactors.

Modeling work on fluid dynamics by FEM in chemical reactors is also not commonly found in the literature. Ranade's (2002) book on reactors computational fluid modeling employs the finite volume method, in the examples and applications presented. Sometimes, commercial packages using the FEM on their built-in routines are employed for the study of chemical reactors models performance (Galante, 2012; Mushtaq, 2014). However, in addition to being proprietary, these routines often focus simulations of chemical reaction media, rather than flow dynamics. Yet, it is possible to verify, in the works by Skrzypacz and Tobiska (2005) and Skrzypacz (2010), a FEM scheme to solve a simple 1-D reactive flow in packed bed reactors. Even though these two studies assume steady flow, BCs are of Dirichlet type and the reaction term is not explicitly solved, the convenience of using FEM in chemical reactors flow modeling is pointed out.

Aims and Objective

Thus, additional motivation exists for the study of concentration fields using FEM, to simulate problems modeled by equation 1 and subjected to unsteady BCs.

Our proposal, and what depicts the main contribution of this work, is to use an outlet BC in the form of a material derivative, directly representing the concentration gradient or the species flux time dependence, an usual feature for such models.

To the authors' knowledge, no analytical solution considering a material derivative as the outlet BC was yet constructed. So, a computer code prototype is developed in MATLAB, through a semi-discrete formulation with GFEM and implicit FD scheme for the simulations. The inlet, or upstream, unsteady BC behavior is assumed either as time periodic, or as pulse functions, providing a variable condition. At the outlet, or downstream, to better represent the equilibrium condition among diffusion, advection and reaction in unsteady conditions, the outlet flux is evaluated by the species concentration material derivative.

MATHEMATICAL FORMULATION

Considering the objectives of the present study, of addressing isothermal reactive flows, an average hydrodynamic field is assumed, so turbulence models are not introduced in the evolution equations. We emphasize that averaging the concentration field along one of the three directions, in order to construct 2-D models, requires that reactants or pollutants be mixed at a much faster rate than the reaction rate, as in the microfluid idealization (Levenspiel, 1999).

The reaction term in equation 1 may considerably vary, depending on the process. For simplicity, it was decided to analyze only a first order reaction model and the diffusion tensor was considered constant. The transport equation then becomes:

$$\frac{\partial C}{\partial t} = -\overline{u}_i \frac{\partial C}{\partial x_i} + D_{ij} \frac{\partial^2 C}{\partial x_i \partial x_j} - kC \tag{5}$$

with initial condition given by:

$$C(x_i, 0) = 0 \tag{6}$$

BCs used at the inlet or upstream are prescribed in one of the two forms below:

$$\left.\begin{aligned} C_{inj}(0,y,t) &= 0, & t &\neq \mathrm{n}\tau \\ C_{inj}(0,y,t) &= C^*_{inj}, & t &= \mathrm{n}\tau \end{aligned}\right\} \tag{7}$$

in order to represent short injections at arbitrary times nτ, or to represent a periodic injection, we assume :

$$C_{inj}(0,y,t) = C_I\,(1 + \cos m\pi t) \tag{8}$$

where C_I is the mean amplitude of the species concentration at the inlet. In equations 7-8, the y coordinate dependence is applicable to 2-D flows and may represent the injection in part or along all its length.

As already mentioned, analytical solutions for this kind of problem exist and will be used in order to validate numerical results. These solutions assume either prescribed or Neumann's outlet BCs mostly at semi-infinite domains. Moreover, even the solutions for finite domains that accept one or other of those BC are subjected to criticism (Ziskind, 2011).

Equation 5 is solved by a FEM scheme, with a Galerkin formulation. So, a weighted residual statement of that equation reads:

$$\int_\Omega \left(\frac{\partial C}{\partial t} + \bar{u}_i \frac{\partial C}{\partial x_i} - D_{ij} \frac{\partial^2 C}{\partial x_i \partial x_j} + kC \right) w d\Omega = 0 \tag{9}$$

By applying the divergence theorem to the third term of the above equation, and substituting the result in equation 9, the following weak form is obtained:

$$\int_\Omega \left(w\frac{\partial C}{\partial t} + w\bar{u}_x.\frac{\partial C}{\partial x} + w\bar{u}_y.\frac{\partial C}{\partial y} + D_x\frac{\partial w}{\partial x}.\frac{\partial C}{\partial x} + D_y\frac{\partial w}{\partial y}.\frac{\partial C}{\partial y} + kwC \right) d\Omega =$$

$$= \int_\Gamma w\left(n_x.D_x\frac{\partial C}{\partial x} + n_y.D_y\frac{\partial C}{\partial y} \right) d\Gamma \tag{10}$$

where $n_x = \vec{n}\cdot\vec{e}_x$, $n_y = \vec{n}\cdot\vec{e}_y$ and $\Gamma = \Gamma_{in} \cup \Gamma_1 \cup \Gamma_2 \cup \Gamma_{out}$.

Γ_1 and Γ_2 represent lateral surfaces and the related fluxes are zero. Γ_{in}, by its turn, represents the inlet boundary, subjected to specified, but time-dependent, BCs, as given by equations 7-8. In this case, the weight functions are zero for Γ_{in}, implying that the surface integral is only evaluated along Γ_{out}.

For the outlet surface, we can assume that:

$$\vec{n} = \vec{e}_x \tag{11}$$

and, therefore, the r.h.s. of equation 10 becomes:

$$\int_{\Gamma_{out}} w\left(D_x \frac{\partial C}{\partial x} \right) d\Gamma_{out} \tag{12}$$

By looking again at equation 12, it can be verified that the weak formulation boundary term represents the species flux by Fick's Law. Yu and Singh (1995) sustain that this formulation should only be applied to situations where there are exclusively diffusion fluxes at the outlet boundary. But in the problems under consideration, advection effectively occurs at the outlet, and must be taken into account in the BC expression.

In fact, there are cases where gradients normal to the outlet surface are zero, bringing the formulation back onto consistency, even in presence of convection because it eliminates the surface integral. Again considering equation 12:

$$\left.\frac{\partial C}{\partial x}\right|_{\Gamma_{out}} = 0 \Rightarrow \int_{\Gamma_{out}} w\left(D_x \frac{\partial C}{\partial x}\right) d\Gamma_{out} = 0 \tag{13}$$

We must have in mind that for a developed profile, equation 13 also implies, taking into account equation 5, in:

$$\left.\frac{\partial C}{\partial t}\right|_{\Gamma_{out}} = -kC \tag{14}$$

We emphasize that this condition does not hold when the gradients at the outlet are not zero. It is well known that flow problems involving the transport of chemical species with homogeneous NBC fail to satisfy the conservation law for species concentrations within the domain (Golz and Dorroh, 2001). In particular, prescribed constant outlet fluxes also do not lead to correct description of time-dependent problems.

So, for the sake of generality another outlet BC must be assumed. We point out that, in the flows under consideration, the species dispersion is mainly due to vertical and transverse velocity gradients, while molecular and turbulent diffusions are generally negligible (Launay et al., 2015). So, adding the advection term to equation 14, one has:

$$\left.\left(\frac{\partial C}{\partial t} + \bar{u}_i . \frac{\partial C}{\partial x_i}\right)\right|_{\Gamma_{out}} = -\left. kC\right|_{\Gamma_{out}} \tag{15}$$

Equation 15 is in fact a nonhomogeneous material derivative that automatically evaluates the spatial gradients at the outlet boundary. We propose to term it *Material Derivative Boundary Condition*, or MDBC, as previously mentioned.

Assuming that, at the boundary Γ_{out}, $\vec{u} = \vec{n}U$, where $\overline{U} = \sqrt{\bar{u}_x^2 + \bar{u}_y^2}$, then, equation 15 can be expressed as:

$$\left.\left(\frac{\partial C}{\partial t} + \overline{U} n_i \frac{\partial C}{\partial x_i}\right)\right|_{\Gamma_{out}} = -\left. kC\right|_{\Gamma_{out}} \tag{16}$$

where: $n_i = \vec{n} \cdot \vec{e}_i$

Following, combining equations 10, 12 and 16, it is possible to write:

$$\int_{\Omega}\left(w\frac{\partial C}{\partial t}+w\bar{u}_x.\frac{\partial C}{\partial x}+w\bar{u}_y.\frac{\partial C}{\partial y}+D_x\frac{\partial w}{\partial x}.\frac{\partial C}{\partial x}+D_y\frac{\partial w}{\partial y}.\frac{\partial C}{\partial y}+kwC\right)d\Omega==-\int_{\Gamma_{out}}w\frac{D_x}{\overline{U}}\left(kC+\frac{\partial C}{\partial t}\right)d\Gamma_{out} \tag{17}$$

Then, equation 17 is the one to be numerically implemented by GFEM, in order to obtain the species concentration profiles.

The numerical procedure may be tested by comparing the results with existing analytical solutions. In the simplest case of 1-D flow, analytical solutions for continuous and pulse mass injection, are, respectively (O'Loughlin and Bowmer, 1975; Chapman, 1979):

$$\frac{C(x,t)}{C_{inj}}=\frac{1}{2}exp\left(\frac{-kx}{\bar{u}_x}\right)erfc\left[\frac{x-\bar{u}_xt(1+H_x)}{\sqrt{4D_xt}}\right] \tag{18}$$

and:

$$C(x,t)=\frac{M_{inj}}{\sqrt{4\pi D_xt}}exp\left[-kt-\frac{(x-\bar{u}_xt)^2}{4D_xt}\right] \tag{19}$$

where $H_x={}^{2kD_x}\!\big/_{\bar{u}_x^2}$ and M_{inj} is the total mass injected per unit area. And for a 2-D case with pulse injection where there is a transversal diffusion D_y and zero lateral component of velocity (Vilhena and Sefidvash, 1985):

$$C(x,y,t)=\frac{M_{inj}}{4\pi t\sqrt{D_xD_y}}exp-\left[kt+\frac{(x-\bar{u}_xt)^2}{4D_xt}+\frac{y^2}{4D_yt}\right] \tag{20}$$

When the inlet BC is given by equation 8, an one-dimensional analytical solution may be obtained. By following the work of Logan and Zlotnik (1996), it is possible to establish that equation 5 clearly admits a solution of the form:

$$C(x,t)=e^{\hat{\alpha}x+\hat{\beta}t} \tag{21}$$

where $\hat{\alpha}$ and $\hat{\beta}$ are complex valued, thus:

$\hat{\alpha}=\alpha_R+i\alpha_I$ and $\hat{\beta}=\beta_R+i\beta_I$ (22

Then, substituting equations 21-22 in the 1-D form of equation 5, one obtains:

$$\hat{\beta}=-k-\bar{u}_x\hat{\alpha}+D_x\hat{\alpha}^2 \tag{23}$$

Once the periodic BC forces the inlet concentration at a fixed value, $\beta_R=0$ and the solution may be expressed as:

$$C(x,t)=\boldsymbol{R}\left[e^{(\alpha_R+i\alpha_I)x+i\beta_It}\right] \tag{24}$$

where ***R*** means the real part of equation 24 and:

$$\alpha_I=\pm\sqrt{\alpha_R^2-\frac{\bar{u}_x}{D_x}\alpha_R-\frac{k}{D_x}} \text{ and } \beta_I=\pm\sqrt{\alpha_R^2-\frac{\bar{u}_x}{D_x}\alpha_R-\frac{k}{D_x}}(2D_x\alpha_R-\bar{u}_x) \tag{25}$$

Also, considering that the concentration at $x = 0$ cannot take negative values, it is necessary to add a constant forcing, such that this restriction is satisfied, and equation 24 becomes:

$$C(x,t) = C_o + \boldsymbol{R}\left[e^{(\alpha_R + i\alpha_I)x + i\beta_I t}\right] \tag{26}$$

For this constant forcing, obviously $\beta_R = \beta_I = 0$ and therefore, with the use of equation 25:

$$\hat{\alpha}_o = -\frac{\overline{u}_x}{2D_x} \pm \sqrt{\frac{\overline{u}_x^2}{4D_x^2} - \frac{k}{D_x}} \tag{27}$$

what implies in:

$$C_o = \boldsymbol{R}\left[e^{(\hat{\alpha}_o)x}\right] \tag{28}$$

Thus, given k, $\overline{u}_x$ and D_x, as well as an abitrary α_R, the analytical solution may be constructed, employing equations 25-28.

NUMERICAL PROCEDURE

By using the Galerkin formulation, the concentration profile is approximated by:

$$C_{appr}(x_i,t) = \sum_{j=1}^{NN} C_j(t) S_j(x_i) \tag{29}$$

Substituting this approximation into the weak form given by equation 17, where, according to the GFEM, the weight functions are the same as the shape functions (Zienkiewicz and Taylor, 2000), one has:

$$\sum_{j=1}^{NN}\left\{\left(\int_\Omega S_i S_j d\Omega + \frac{D_x}{\overline{U}}\int_{\Gamma_{out}} S_i S_j d\Gamma_{out}\right)\frac{dC_j}{dt} + \int_\Omega\left[S_i\left(\overline{u}_x\frac{\partial S_j}{\partial x} + \overline{u}_y\frac{\partial S_j}{\partial y}\right) + \left(D_x\frac{\partial S_i}{\partial x}\frac{\partial S_j}{\partial x} + \right.\right.\right.$$

$$\left.\left.\left. + D_y\frac{\partial S_i}{\partial y}\frac{\partial S_j}{\partial y}\right)\right]d\Omega C_j + k\left(\int_\Omega S_i S_j d\Omega + \frac{D_x}{\overline{U}}\int_{\Gamma_{out}} S_i S_j d\Gamma_{out}\right)C_j\right\} = 0 \tag{30}$$

where the boundary integral (r.h.s of equation 17) was approximated through:

$$\sum_{j=1}^{NN}\left[\frac{D_x}{\overline{U}}\left(\int_{\Gamma_{out}} S_i S_j d\Gamma_{out}\right)\left(-kC_j + \frac{dC_j}{dt}\right)\right] \tag{31}$$

Equation 30 encompasses a stiffness matrix and a modified mass matrix which is related to the concentration time derivative and the reaction term. It can be put under matrix form as:

$$[\mathrm{M}_1]\left\{\overset{\bullet}{C}\right\}+[\mathrm{K}]\{C\}+k[\mathrm{M}_1]\{C\}=0 \tag{32}$$

where, M1 and K are, respectively, the modified mass and stiffness matrices.

In order to solve equation 32, we employ a numerical scheme, using the Crank-Nicholson Method (Lewis et al., 2005), which reads:

$$\{C_{t+1}\}=\left([\mathrm{M}_1]+\frac{\Delta t}{2}(\mathrm{K}+k[\mathrm{M}_1])\right)^{-1}\left([\mathrm{M}_1]-\frac{\Delta t}{2}(\mathrm{K}+k[\mathrm{M}_1])\right)C_t \tag{33}$$

It must be observed that it is also possible to look for another solution without modifying the original mass matrix, as suggested above. In this case, the use of the Crank-Nicholson scheme on GFEM approximation of equation 10, implies in:

$$\{C_{t+1}\}=\left([\mathrm{M}]+\frac{\Delta t}{2}[\mathrm{K}_1]\right)^{-1}\left[\left([\mathrm{M}]-\frac{\Delta t}{2}[\mathrm{K}_1]\right)C_t+\frac{\Delta t}{2}\left(\{B\}_t+\{B\}_{t+1}\right)\right] \tag{34}$$

where $\{B\}_t$ and $\{B\}_{t+1}$ are the boundary terms arising from the line integral approximation on the r.h.s of equation 10 at times t and $t+1$, [M] is $\sum_{j=1}^{NN}\int_{\Omega} S_i S_j d\Omega$ and [K1] is a modified stiffness matrix, now including the decay term, last on the left term of equation 17, or:

$$\sum_{j=1}^{NN}\left\{\int_{\Omega}\left[S_i\left(\bar{u}_x\frac{\partial S_j}{\partial x}+\bar{u}_y\frac{\partial S_j}{\partial y}\right)+\left(D_x\frac{\partial S_i}{\partial x}\frac{\partial S_j}{\partial x}+\right.\right.\right.$$
$$\left.\left.\left.+D_y\frac{\partial S_i}{\partial y}\frac{\partial S_j}{\partial y}+kS_iS_j\right)\right]d\Omega C_j\right\} \tag{35}$$

In this case, the boundary vectors ($\{B\}_t$ and $\{B\}_{t+1}$) must be evaluated using equation 31. Being dependent on the concentration and its time derivative in past and present time steps, these vectors must be continuously updated, making the numerical scheme for solving equation 33 simpler than the one required for solving equation 34. Thus, we opted for the first scheme.

The code was implemented in MATLAB, taking advantage of its matrix calculation resources. The integrals in equation 30 were evaluated by the Gauss Quadrature (GQ). The solution domain was discretized in regular triangular or quadrangular element meshes by routines within the program, depending on the case run. The program is also capable of performing GQ calculations in diversified number of interval points. Linear shape functions were used throughout this work, so precision of the scheme was controlled by properly refining the mesh.

It is well known that simple GFEM presents numerical oscillations and instabilities in problems where advection is important. So, more elaborated FEM schemes would be required to solve problems with small diffusion coefficients. However, considering that the role of the unsteady BC along with the outlet BC represented by a material derivative were the main aspects to be investigated, this method was employed with restrictions. Aware that some of the major factors causing these issues are improper choice of a time step size and also of element size and shape (Yu and Singh, 1995), we adopted, as a basis for the time step and element size control, respectively, (Chapra and Canale, 2010):

$$\Delta t_i \le \frac{(\Delta x_i)^2}{2D_{x_i} + k.(\Delta x_i)^2} \qquad \text{and} \qquad \Delta x_i \le \frac{2D_{x_i}}{\overline{u}_i} \tag{36}$$

RESULTS AND DISCUSSION

Preliminary Tests

A more detailed look at the analytical solution presented by equation 18 reveals that, actually, the assumed constant upstream BC is not time independent, as it may appear to be in a first glimpse. Assuming unitary injection concentration (C_{inj} = 1.0), the analytical solution result in the plots of Figure 1, obtained for Pe = 5.0 ($\overline{u}_x$ = 1.0; D_x = 4.0; Da = 2.0).

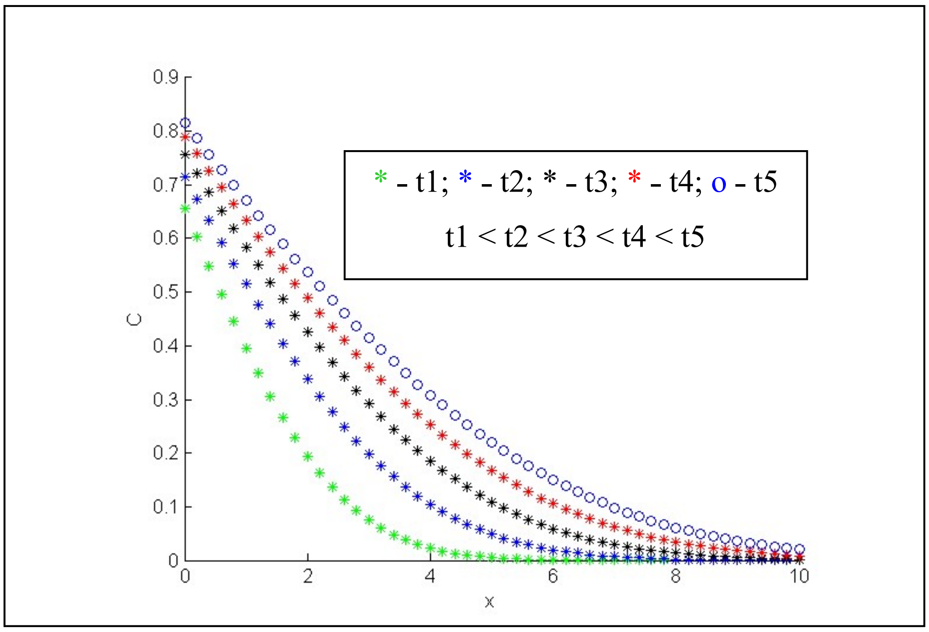


Figure 1. 1-D Plot of Analytical Solution (Equation 18).

As it can be verified, within the stream limits, the BC shows an unsteady profile characterized by the inlet concentration correction due to particular advective and diffusion effects. Obviously, the analytical solution follows the general form of the concentration profile for this kind of problems (Vilhena and Sefidvash, 1985):

$$C(x,t) = C_0(x,t)exp(-kt) \tag{37}$$

where C_o is the corrected species concentration to initial time.

It can be also easily seen, by inspection of equations 19 and 20, that the solution for pulse injections also follows equation 37, in order to correct the inlet concentration values.

So, in order to check the code results, the inlet BCs to be applied at x=0 must carry on the initial shape of the defined concentration, as suggested by Yu and Li (1998). This implies in:

d) for equation 18:

$$C_0(0,t) = \frac{C_{inj}}{2}\left\{ erfc\left[\frac{-\bar{u}_x t(1+H_x)}{\sqrt{4D_x t}} \right] \right\} \tag{38}$$

e) for equation 19:

$$C_0(0,t) = \frac{M_{inj}}{\sqrt{4\pi D_x t}} exp\left[-kt - \frac{(-\bar{u}_x t)^2}{4D_x t} \right] \tag{39}$$

and:

f) for equation 20:

$$C_0(0,y,t) = \frac{M_{inj}}{4\pi t\sqrt{D_x D_y}} exp\left[-kt - \frac{(-\bar{u}_x t)^2}{4D_x t} - \frac{y^2}{4D_y t} \right] \tag{40}$$

Having that in mind, one can apply equations 38-40 to the MATLAB code and compare the results with the analytical solutions for constant and pulse injection cases.

In the following Figures 2-3, conditions for Pe = 5.0 are the same as for Figure 1; for Pe = 50 are: $\bar{u}_x = 10$, $D_x = 4.0$ and $k = 1.0$; for Pe = 200 are: $\bar{u}_x = 10$, $D_x = 1.0$ and $k = 1.0$, resulting in the same Damköhler Number (2.0) for all cases. For the tests with equation 20, which admits a lateral component of diffusion, D_y was set equal to 0.2 and its 1-D plot (graph C of Figure 2) represents the centerline concentration profile ($y = 0.0$).

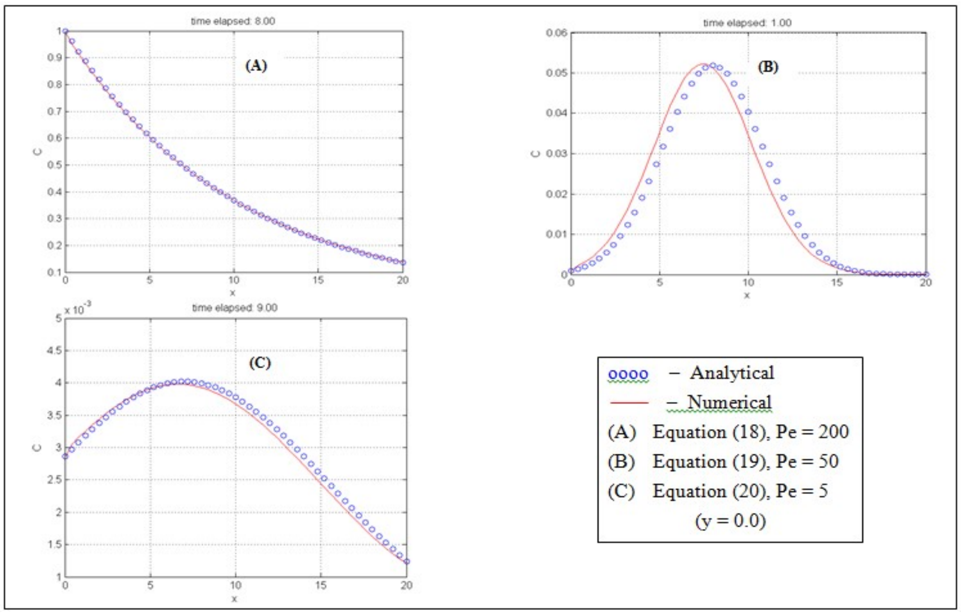


Figure 2. 1-D Analytical and Numerical Solution of Equations 18-20 Cases.

The numerical solution of equation 5, for the periodic inlet BC (equation 8), may be compared with the 1-D analytical solution constructed from equations 25-28 through a plot extracted from the centerline concentration profile. Figure 4 shows the outcome for Pe = 100, where $\bar{u}_x = 5.0$, $D_x = 1.0$ and $k = 0.1$, implying in Da = 0.4.

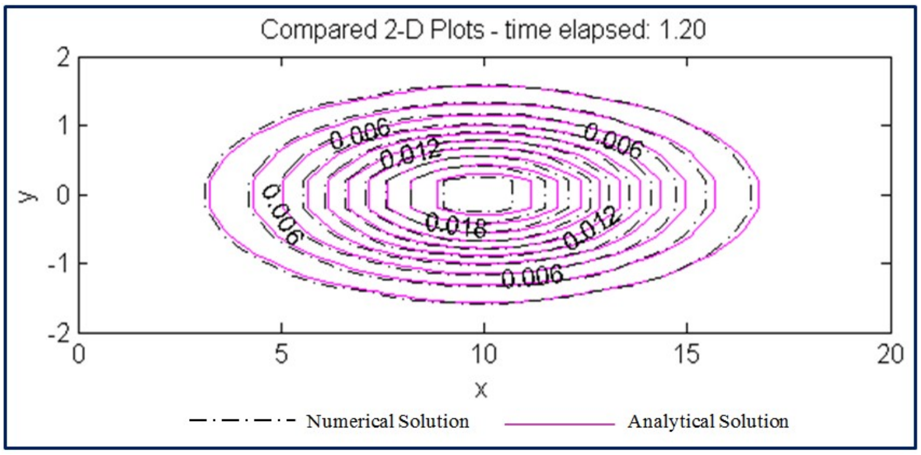


Figure 3. 2-D Analytical and Numerical Solution of Equation 20 Case. (1250 Elements Mesh; GQ 9 points; Pe = 50)

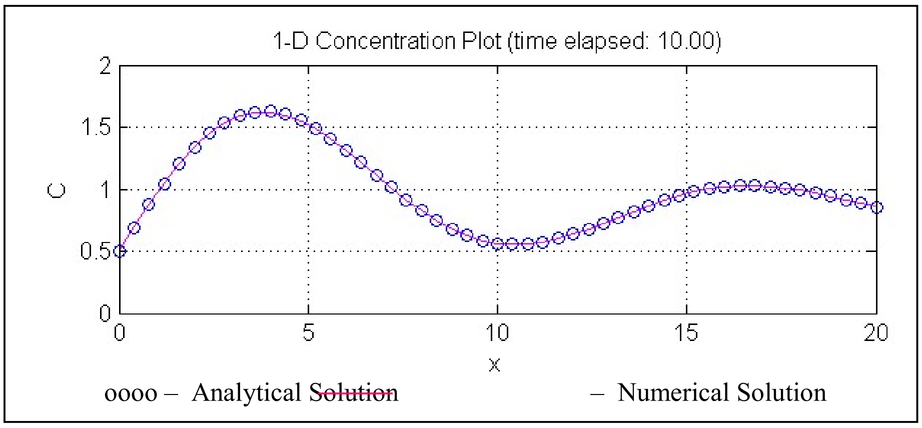


Figure 4. 1-D Analytical and Numerical Solutions of Equation 5 with periodic inlet BC. (1250 Elements Mesh; GQ 9 points).

In order to obtain the plots of Figures 2 to 4, we run the code and then compared the results with the analytical solution correspondent to the time run. Numerical calculation was performed, respecting the stability restrictions posed by equations 36. The plots show good agreement between analytical and numerical solutions even for high Péclet Numbers.

It is possible to observe a better agreement between analytical and numerical solutions for the continuous injection case (equation 15), as shown in plot A of Figure 2. The plots B and C of Figure 2 (equations 19 and 20) and the plot of Figure 3 (equation 20), show that the numerical curves are slightly delayed compared to the exact solutions. This delay results from the fact that the discrete time integration cannot completely follow the instant moment of mass release (Lee and Seo, 2010).

Comparing Analytical and Numerical Solutions

Figure 5 compares simulated concentration profiles for sorted conditions, such as Pe = 5 ($\overline{u}_x = 1.0$; $D_x = 4.0$; $k = 0.1$), Pe = 25 ($\overline{u}_x = 5.0$; $D_x = 4.0$; $k = 0.1$) and Pe = 100 ($\overline{u}_x = 5.0$; $D_x = 1.0$; $k = 0.1$). In order to obtain the plots, we solved equation 5 subjected to a time periodic inlet BC (equation 8), changing the outlet BC type. First, we employed an EBC arbitrarily set to a given constant value, then, we employed a homogeneous NBC and last, our proposed MDBC. Since the meshes used were the same in all simulations, we compared the centerline nodes values obtained, plotting the concentrations differences (Dif C).

As we can see, profiles obtained when the adopted outlet condition is either EBC or the homogeneous NBC, compared to those obtained by the adoption of the MDBC, concentrate larger differences around the exit.

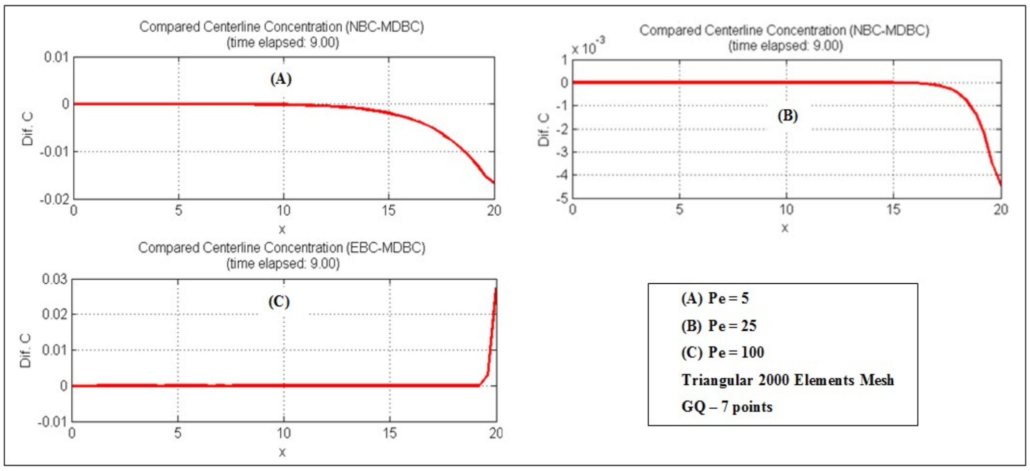


Figure 5. Centerline Concentration Profile Differences – Numerical Solutions.
(Inlet BC: Equation 8; Outlet EBC = 0.5; Outlet NBC: Equation 13; Outlet MDBC: Equation 16).

In order to check the validity of the above proposition, we numerically evaluated concentrations 1-D profiles for various flow and reaction parameters. The results were compared to the analytical solution and analyzed by the Root-Mean-Square Deviation (RMSD), or:

$$\textbf{RMSD} = \sqrt{\frac{\sum_{i=1}^{nd}\left(C_i - C_i^a\right)^2}{nd}} \tag{41}$$

where C_i^a is the analytical solution at node *i* for a given total number of nodes *nd* at the exit region.

Table1. RMSD between 1-D Analytical and Numerical Solutions.

Pe = 100		RMSD			
Δx	Δt	Da	An. - EBC	An. - NBC	An. - MDBC
0.2	0.02	0.1	0.8193	0.0315	0.0226
		1.0	0.2640	0.1839	0.1744
		2.0	0.0745	0.0044	0.0022
Pe = 50		RMSD			
0.2	0.005	0.1	0.8628	0.0098	0.0041
	0.05	1.0	0.4982	0.0106	0.0032
		2.0	0.1312	0.0809	0.0798
Pe = 25		RMSD			
0.2	0.02	0.1	0.8846	0.0777	0.0537
0.4	0.01	1.0	0.4010	0.0676	0.0679
0.2	0.02	2.0	0.1476	0.0387	0.0259
Pe = 5		RMSD			
0.2	0.05	0.1	0.6155	0.0275	0.0191
	0.2	1.0	0.5672	0.0071	0.0072
	0.1	2.0	0.0880	0.0178	0.0034

(Inlet BC: Equation 8; Outlet EBC = 0.0; Outlet NBC: Equation 13; Outlet MDBC: Equation 16).

Outcome Analysis

We observe that the numerical solutions with outlet EBC provide the poorest approximations in all Péclet and Damköhler Numbers considered and that MDBC solutions result in better approximations than NBC in almost all cases. This is possibly due to the fact that MDBC better captures specific features of the flow because it encompasses, in its formulation, physical effects of the problem which are not present in the usual types of BCs.

Results on Table 1 also point at examples where the advantages of using MDBC instead of homogeneous NBC are not clear. Such situations arise from particular flow conditions that imply in very small concentration gradients at the outlet, as consequence of Péclet and Damköhler Numbers combinations. These cases approach patterns that can be treated conveniently by the homogeneous NBC (equation 3) and so, when we compare the outcomes obtained both with the use of NBC and MDBC, we verify analogous deviation from the analytical solution. However, these are special cases of the problem and the use of the MDBC for more general formulations is established.

2-D Simulation Results

Having in mind the satisfactory results obtained in the tests, we further used the code to investigate the behavior of 2-D systems. Velocities and diffusion constants were chosen as close as possible to real configurations.

For instance, Figure 6 shows the results of 2-D and 1-D simulations under conditions such that the inlet BC is the periodic concentration oscillation given by equation 8, lateral components of velocity and diffusivity are ten times smaller than the longitudinal components ($\bar{u}_x = 5.0$, $\bar{u}_y = 0.5$, $D_x = 1.0$, $D_y = 0.1$ and $k = 0.1$), implying in Pe = 100 and Da = 0.4.

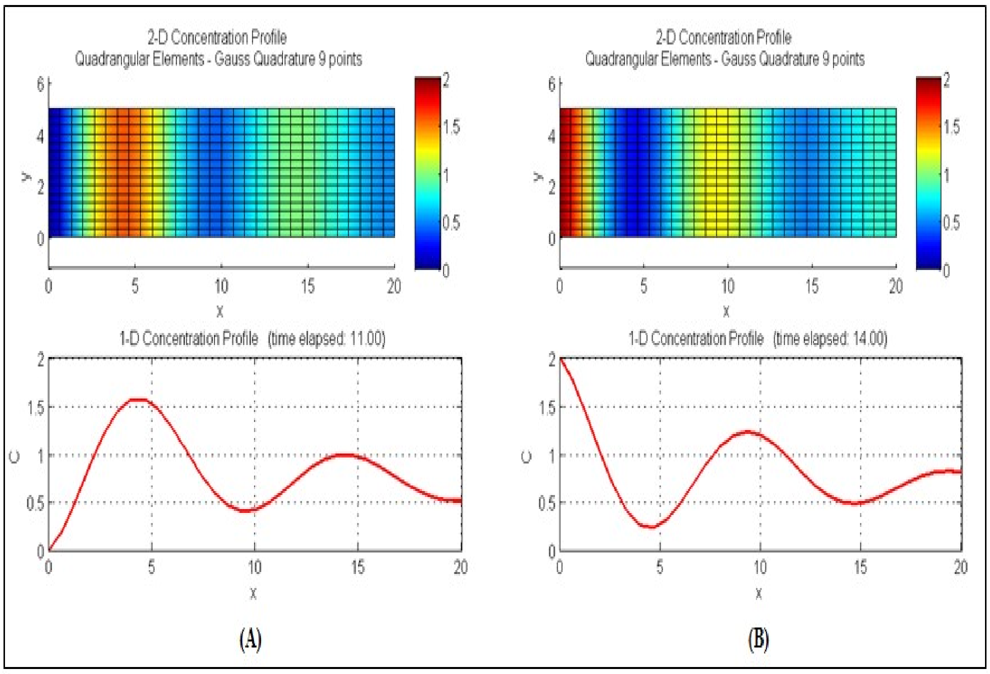


Figure 6. Concentration Profile for Decaying Species.
(900 Elements Mesh - Inlet BC: Equation 8; Outlet BC: Equation 16)

In this case, corresponding to a high Pe, convective transport plays a major role overcoming diffusion transport and reaction decay. Parts A and B of Figure 6 show the oscillatory behaviour of the concentration profile along the domain at different time values for the concentration along all the domain. We also note the variable outlet concentration values that would not properly be captured by EBCs and possibly NBCs.

Figure 7 shows the outlet concentrations for Pe = 10 and Da = 0.4 ($\overline{u}_x$ =0.5; $\overline{u}_y$ =0.05; D_x =1.0; D_y =0.1; k=0.01), subject to the same BCs, implying a more important role for diffusive transport. In addition, smaller flow rates allow the chemical reaction to further evolve as the convective transport takes place. Following, the oscillatory behavior of the inlet concentration is damped before reaching the domain outlet and the solution approaches the typical shape of pure diffusive transport problems subjected to oscillatory BC, known as *periodic steady-state* (Bird et al, 2002).

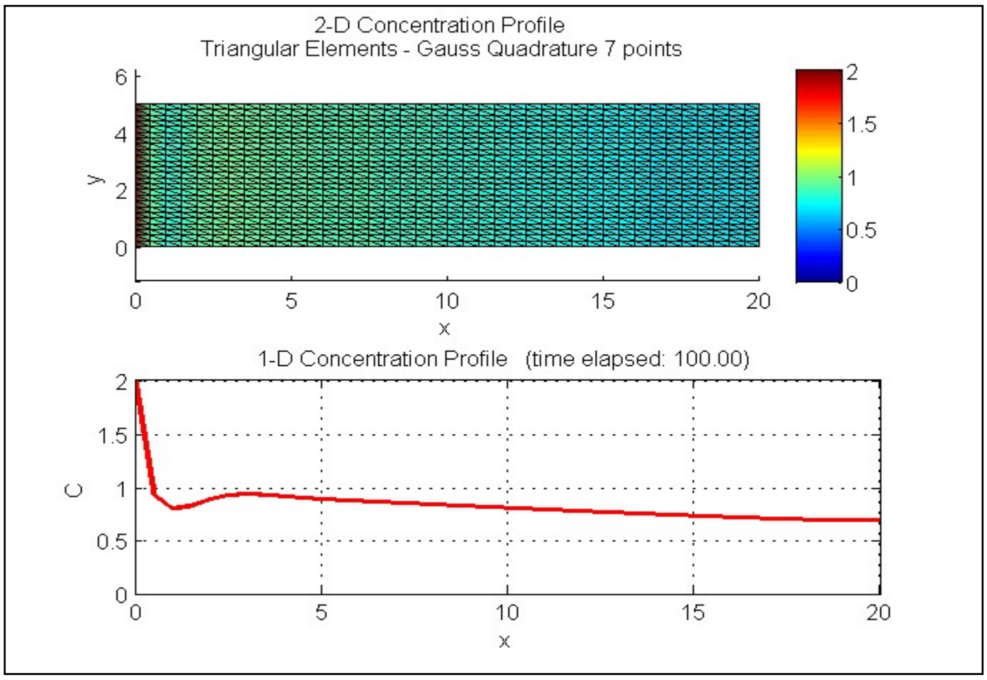


Figure 7. Concentration Profile for Decaying Species.
(2000 Elements Mesh - Inlet BC: Equation 8; Outlet BC: Equation 16)

When equations 7 are applied as the inlet BC, resulting in pulse injection of time-dependent concentrations, the code shows the concentration profiles approaching the oscillatory profile as the interval time between each injection becomes shorter (part A of Figure 8), or the pulse injection profile (part B of Figure 8), in a Gaussian shape, as it becomes larger.

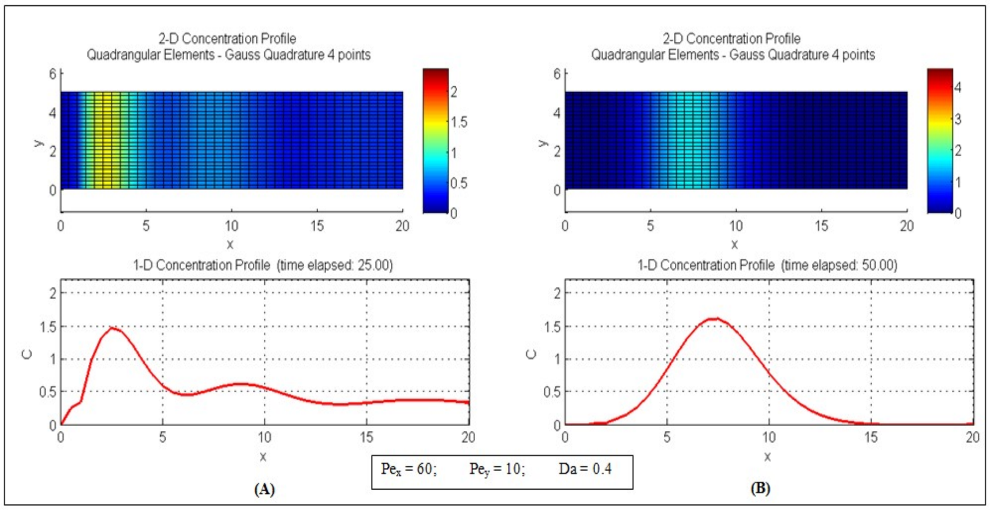


Figure 8. Concentration Profile for Decaying Species.
(1000 Elements Mesh - Inlet BC: Equation 7; Outlet BC: Equation 16)

The code is able to simulate 2-D configurations. including flow predictions when the velocities profiles are steady but dependent on the spatial coordinates such that $\overline{u}_x = \overline{u}_x(y)$ and $\overline{u}_y = \overline{u}_y(x)$. For example, if a steady parabolic profile is considered for the longitudinal velocity (equation 42), for the same other parameters as those of Figure 4, Figure 9 is obtained:

$$\overline{u}_x = 5.0y - y^2 \tag{42}$$

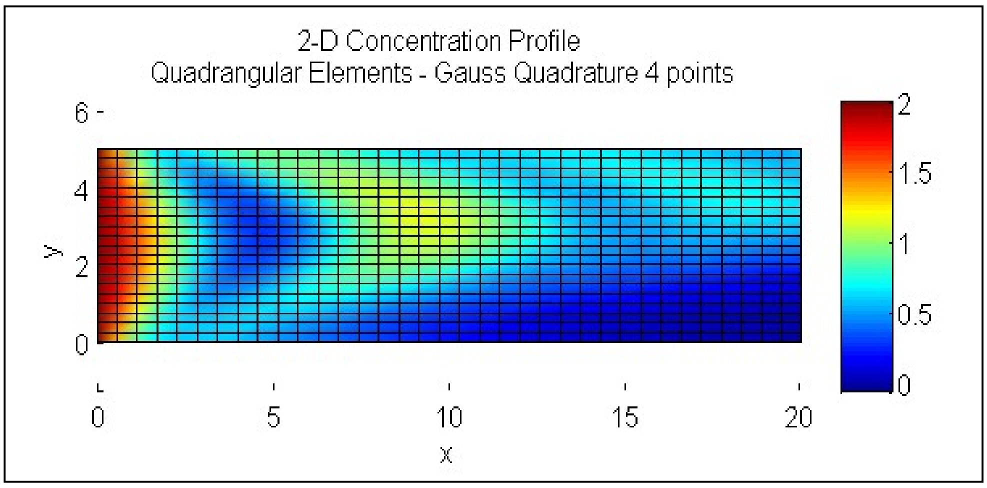


Figure 9. Concentration Profile for Decaying Species – Parabolic Longitudinal Velocity.
(700 Elements Mesh - Inlet BC: Equation 8; Outlet BC: Equation 16)

Figure 9 depicts the evolution of the species cloud deformed due to the existence of lateral components of velocity and diffusion. But the mass injection occurs uniformly at the inlet cross section area, a condition most found in chemical reactors or in small channels.

So, in order to demonstrate the code ability to simulate conditions more likely to happen in large watercourses, we modify the inlet BC as follows. Considering that in 2-D analysis the inlet may also be dependent on y (equation 8) we are able to obtain results:

a) for centered pointsource injection:

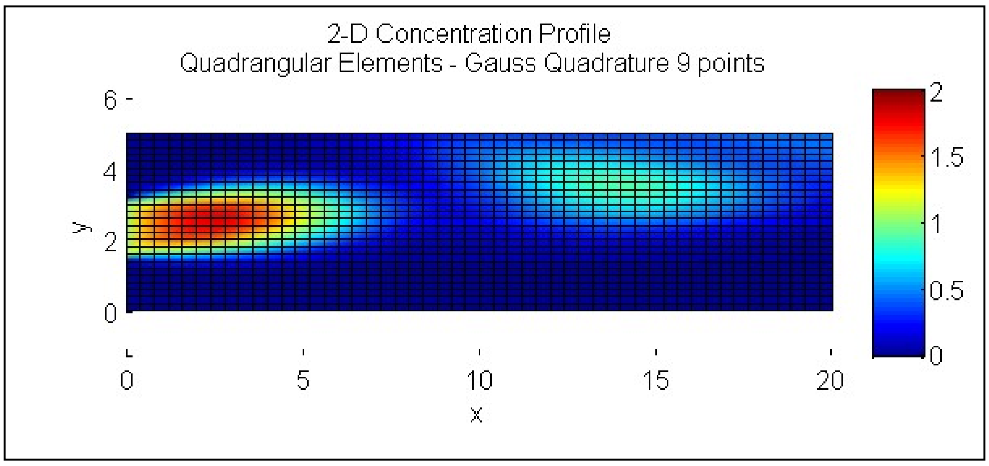


Figure 10. Concentration Profile for Decaying Species – Left centerline injection. (1250 Elements Mesh - Inlet BC: Equation 8; Outlet BC: Equation 16)

b) for right margin (left bottom) pointsource injection:

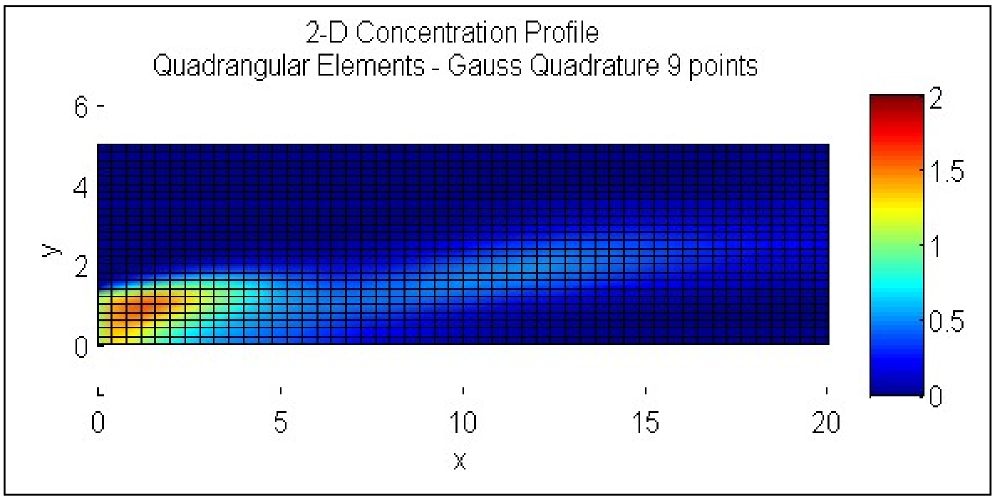


Figure 11. Concentration Profile for Decaying Species – Bottom left injection. (1250 Elements Mesh - Inlet BC: Equation 8; Outlet BC: Equation 16)

c) for left margin (upper left) pointsource injection:

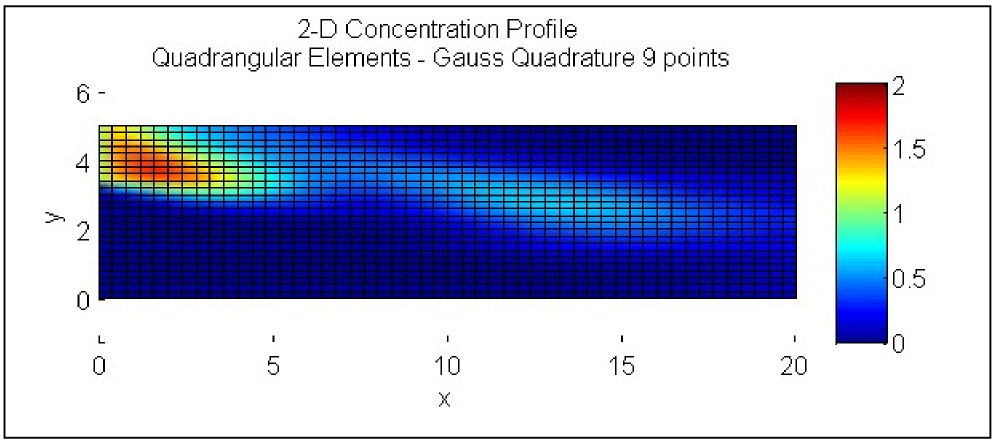


Figure 12. Concentration Profile for Decaying Species – Upper left injection. (1250 Elements Mesh - Inlet BC: Equation 8; Outlet BC: Equation 16)

Figures 10 and 11 are obtained from the same parameters as those for Figure 9 and in Figure 12 the lateral component of the velocity is set from the upper margin downwards, assuming the negative of Figure 9 value for this same component.

CONCLUSION

In transient reactive flow problems subjected to unsteady BC the main issue is to achieve physical coherence in constructing the model to be solved. Some analytical solutions of this class of problems are found in the literature which, though being parabolic, usually assume the outlet BC in the form of a constant concentration or of a given concentration gradient.

As indicated by Piasecki and Katopodes (1997), simulations presented in this work confirmed that oscillatory inlet conditions result in time-dependent concentrations at the outlet, that cannot be accounted for by EBCs and NBCs. Also, NBCs may not represent the total equilibrium flux at the outlet (Yu and Singh, 1995), leading to physically incomplete models that could perform imprecise profile estimation.

A new procedure was then proposed, by which a material derivative is considered as the outlet BC. Our results show that these BCs provide a better picture of the process, updating the outlet equilibrium concentration.

A MATLAB code was developed with a numerical scheme subjected to prescribed stability restrictions (equations 36), using a semi-implicit GFEM scheme. Good agreement was obtained between simulations and existing analytical solutions, as can be seen on Figures 2 to 4 and Table 1. It is also shown in Table 1 and Figure 5 comparisons of numerical solutions using EBC, homogeneous NBC and the proposed MDBC, evidencing the positive aspects of applying the material derivative as the outlet BC. Following, 2-D simulations were then performed in rectangular channels, assuming fully developed velocity profiles.

The code features a certain flexibility for automatically generating regular triangular and quadrangular meshes that could be selected to the applicable case. There was also the option of changing the number of GQ points to evaluate the model integrals, known to slightly affect the computational time.

Several simulations were run on a i5 CPU notebook, limited to a maximum of 2000 element meshes, all requiring few minutes to run, showing that even more refined meshes could be used while keeping CPU times within acceptable limits. Our tests indicate that the numerical scheme is sufficiently tested to be implemented in codes written in lower level languages.

The use of FEM in reactive flows simulation was reinforced and, finally, a further improvement could be made in the code by future works, in the sense of adopting more elaborated FEM formulations, involving a SUPG or other more advanced stabilization technique, so to combine the advantages of more stable schemes with the proposed adoption of the MDBC.

NOMENCLATURE

C	section-averaged species concentration

C_{appr}	approximated concentration given by the FEM formulation
C_{inj}	injected averaged concentration
Dx, D_y	averaged diffusion coefficient in the direction of the respective coordinate axis
Da	Damköhler Number
k	reaction constant or pollutant decay constant
m, n	arbitrary integers 1, 2, 3 …
NN	number of nodes in the finite element mesh
Pe	Péclet Number
r	reaction term
$S_j(x_i)$	shape function
t	time
$\overline{u}_i$	averaged flow velocity along coordinate x_i
w	arbitrary weight function
x_i	coordinate in an arbitrary direction i
Γ	control surface
Γ_s	arbitrary boundary on surfaces
τ	arbitrary time between injections
Ω	control volume

ACKNOWLEGEMENTS

The authors wish to thank the Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ) for its support to this research under the program number E-26/111.079/2013.

REFERENCES

Aral, M. M. and Liao, B. Analytical Solution for Two-Dimensional Transport Equation with Time-Dependent Dispersion Coefficients., J. Hydrol. Engrg., 1, nº. 1, 20-32 (1996).

Bird, R. B., Stewart, W. E. and Lightfoot, E. N., Transport Phenomena, 2nd ed. John Wiley & Sons, New York (2010).

Chapman, B. M., Dispersion of Soluble Pollutants in Non-uniform Rivers – I. Theory, J. Hydrol., 40, 139-152 (1979).

Chapra, S.C and Canale, R. P., Numerical Methods for Engineers, 6th ed. Mc Graw-Hill, New York (2010).

Chen, J. S. and Liu, C. W., Generalized Analytical Solutions for Advection-Dispersion Equation in Finite Spatial Domain with Arbitrary Time-Dependent Inlet Boundary Condition. Hydrol. Earth Syst. Sci., 15, 2471-2479 (2011).

Czernuszenko, W., Dispersion of Pollutants in Rivers, Hydr. Sci. J., 32, nº. 1, 59-67 (1987).

Galante, R. L., Modeling and Simulation of a Continuous Tubular Reactor for the Biodiesel Production (in Portuguese), D.S. Thesis, University of Santa Catarina (2012).

Galeão, A. C., Almeida, R. C., Malta, S. M. C. and Loula, A. F. D., Finite Element Analysis of Convection Dominated Reaction-Diffusion Problems, Appl. Num. Math, 48, 205-222 (2004).

Goltz, W. J. and Dorroh, J. R. The Convection-Diffusion Equation for a Finite Domain with Time Varying Boundaries, Appl. Math. Lett., 14, 983-988 (2001).

John, V. and Schmeyer, E., Finite Element Methods for Time-Dependent Convection-Diffusion-Reaction Equations with Small Diffusion, Comp. Meth. Appl. Mech. Engrg, 198, 475-494 (2008).

Kachiashvili, K., Gordeziani, D, Lazarov, R. and Melikdzhanian, D., Modelling and Simulation of Pollutants Transport in Rivers, Appl. Math. Mod., 31, 1371-1396 (2007).

Konzen, P. H., De Bortoli, A. L. and Thompson, M., Finite Element Method Applied to the Solution of a Convective-Diffusive-Reactive Flow, In: XXX Congresso Nacional de Matemática Aplicada e Computacional – XXX CNMAC (Annals), Florianópolis (2007). Available at: http://www.sbmac.org.br/eventos/cnmac/xxx_cnmac/PDF/294.pdf.

Launay, M., Le Coz, J., Camenen, B., Walter, C., Angot, H., Dramais, G., Faure, J.-B. and Coquery, M., Calibrating Pollutant Dispersion in 1-D Hydraulic Models of River Networks, J. Hydro-env. Res., 9, 120-132 (2015).

Lee, M. E. and Seo, I. W., Analysis of Pollutant Transport in the Han River with Tidal Current Using a 2D Finite Element Model, J. Hydro-env. Res., 1, n^{o}.1, 30-42 (2007).

Lee, M. E. and Seo, I. W., 2D Finite Element Pollutant Transport Model for Accidental Mass Release in Rivers, KSCE J. Civ. Eng., 14, n^{o}.1, 77-86 (2010).

Levenspiel, O., Chemical Reaction Engineering, 3rd ed. John Wiley & Sons, New York (1999).

Lewis, R. W., Nithiarasu, P. and Seetharamu, K. N., Fundamentals of the Finite Element Method for Heat and Fluid Flow. John Wiley & Sons, Chichester (2005).

Logan, D. L. A First Course in the Finite Element Method - 4th Edition. Thomson, Ontario (2007).

Logan, J. D. Solute Transport in Porous Media with Scale-Dependent Dispersion and Periodic Boundary Conditions, J. Hydr., 184, n^{o}. 3, 261-276 (1996).

Logan, J. D. and Zlotnik, V. The Convection-Diffusion Equation with Periodic Boundary Conditions, Appl. Math. Lett., 8, n^{o}. 3, 55-61 (1995).

Mushtaq, F., Analysis and Validation of Chemical Reactors Performance Models Developed in a Commercial Software Platform, M.S. Thesis, KTH School of Industrial Engineering and Management, Stockholm (2014).

O'Loughlin, E. M. and Bowmer, K. H., Dilution and Decay of Aquatic Herbicides in Flowing Channels, J. Hydrol., 26, 217-235 (1975).
Pérez Guerrero, J. S., Pontedero, E. M., van Genuchten M. Th. and Skaggs, T. H., Analytical Solutions of the One-Dimensional Advection-Dispersion Solute Transport Equation Subject to Time-Dependent Boundary Conditions. Chem. Eng. J., 221, 487-491 (2013).

Piasecki, M. and Katopodes, N. D., Control of Contaminant Releases in Rivers I: Adjoint Sensitivity Analysis, J. Hydrau. Eng., 123, n^o. 6, 486-492 (1997).

Ranade, V. V., Computational Flow Modeling for Chemical Reactor Engineering. Academic Press, San Diego (2002).

Skrzypacz, P. and Tobiska, L., Finite Element and Matched Asymptotic Expansion Methods for Chemical Reactor Flow Problems, Proc. Appl. Math. Mech., 5, 843-844 (2005).

Skrzypacz, P., Finite Element Analysis for Flow in Chemical Reactors, Dr. Rer. Nat. Dissertation, Otto von Guericke Universität, Magdeburg (2010).

van der Perk, M., Soil and Water Contamination, 2nd ed. CRC/Balkema, Leiden (2013).

van Genuchten, M. Th. and Alves, W. J. Analytical Solutions of the One-Dimensional Convective-Dispersive Solute Transport Equation, US Dept. of Agric. Tech. Bull, no. 1661 (1982).

Vilhena, M.T. and Leal, C. A., Dispersion of Non-Degradable Pollutants in Rivers, Int. J. Appl. Radiat. Isot., 32, 443-446 (1981).

Vilhena, M.T. and Sefidvash, F., Two-dimensional Treatment of Dispersion of Pollutants in Rivers, Int. J. Appl. Radiat. Isot., 36, n^o. 7, 569-572 (1985).

Yu, F. X. and Singh, V. P., Improved Finite Element Method for Solute Transport, J. Hydraul. Eng., 121, n^o. 2, 145-158 (1995).

Yu, T.S. and Li, C. W., Instantaneous Discharge of Buoyant Fluid in Cross-flow, J. Hydraul. Eng., 124, n^o. 1, 1161-1176 (1998).

Zienkiewicz, O. C. and Taylor, R. L., The Finite Element Method, Vol 1: The Basis, 5th ed. Butterworth-Heinemann, Oxford (2000).

Ziskind, G., Shmueli, H. and Gitis, V., An Analytical Solution of the Convection-Dispersion-Reaction Equation for a Finite Region with a Pulse Boundary Condition, Chem. Eng. J., 167, 403-408 (2011).

Brazilian Journal of Chemical Engineering

Decision Letter (BJCE-2016-0044.R2)

From: jefferson.gomes@abdn.ac.uk

To: ade_oliveira@hotmail.com

CC:

Subject: Brazilian Journal of Chemical Engineering - Decision on Manuscript ID BJCE-2016-0044.R2

Body: 12-Jul-2016

Dear Mr. DE OLIVEIRA FILHO:

It is a pleasure to accept your manuscript entitled "SIMULATION OF SPECIES CONCENTRATION DISTRIBUTION IN REACTIVE FLOWS WITH UNSTEADY BOUNDARY CONDITIONS" in its current form for publication in the Brazilian Journal of Chemical Engineering. The comments of the reviewer(s) who reviewed your manuscript are included at the foot of this letter.

We are starting the next steps for publication, the English proofreading and the preparation of the galley proofs (please note that these steps may take some additional time because there is a line of accepted articles being processed on a first-come-first-served basis). Please wait for a future contact from the Editor-in-Chief or from our publication office to verify the outcome of these steps.

Thank you for your fine contribution. On behalf of the Editors of the Brazilian Journal of Chemical Engineering, we look forward to your continued contributions to the Journal.

Sincerely,
Dr. Jefferson Gomes
Associate Editor, Brazilian Journal of Chemical Engineering
jefferson.gomes@abdn.ac.uk

Entire Scoresheet:

Date Sent: 12-Jul-2016

Close Window