```
%{
clear all;
close all;
clc;
HX =  1; HY =  1;
MX = 40; MY = 40;
alpha  = 1;
sourceFun = @(x,y,t,alpha) (-2*exp(-t^2)*(2*pi*alpha*y.*cos(2*pi*x).*cos(2*pi*y) -
2*pi*alpha*x.*sin(2*pi*x).*sin(2*pi*y) + t*x.*y.*cos(2*pi*y).*sin(2*pi*x) -
4*pi^2*alpha*x.*y.*cos(2*pi*y).*sin(2*pi*x)));
u0 = @(x,y)(x.*y.*sin(2*pi*x).*cos(2*pi*y));
tStop = 2;
DFL = 1.0;
reportStep = 100;
ua = @(x,y,t)(x.*y.*sin(2*pi*x).*cos(2*pi*y).*exp(-t^2));

bcType.west = 'D';
bcType.east = 'D';
bcType.north = 'N';
bcType.south = 'N';

bcFun.west  = @(t,y,alpha) (zeros(size(y)));
bcFun.east  = @(t,y,alpha) (zeros(size(y)));
bcFun.south = @(t,x,alpha) (-alpha*x.*sin(2*pi*x).*exp(-t^2));
bcFun.north = @(t,x,alpha) ( alpha*x.*sin(2*pi*x).*exp(-t^2));

xMeshType = 'uniform';
yMeshType = 'uniform';

[x,xc,side_x,delt_x,y,yc,side_y,delt_y,dt,nSteps,t,u]=PRADI(HX,HY,MX,MY,alpha,
sourceFun,bcType,bcFun,u0,tStop,DFL,reportStep,ua,xMeshType,yMeshType);
%}
function [x,xc,side_x,delt_x,y,yc,side_y,delt_y,dt,nSteps,t,u]=PRADI(HX,HY,MX,MY,alpha,
sourceFun,bcType,bcFun,u0,tStop,DFL,reportStep,ua,xMeshType,yMeshType)

[x,xc,side_x,delt_x] = meshGen(HX,MX,xMeshType);
[y,yc,side_y,delt_y] = meshGen(HY,MY,yMeshType);

dt = DFL*(min(side_x).^2 + min(side_y).^2)/alpha;

nSteps = ceil(tStop/dt);

t = 0;

u = nan(MX+2,MY+2);
for i=2:MX+1
    for j=2:MY+1
        u(i,j) = u0(xc(i-1),yc(j-1));
    end;
end;
u = ghostCells(u,bcType,bcFun,t,xc,yc,delt_x,delt_y,alpha);


hf=figure;
[Y,X]=meshgrid([yc(1)-delt_y(1);yc;yc(end)+delt_y(end)],[xc(1)-delt_x(1);xc;xc(end)
```

```
+delt_x(end)]);
h=surf(X,Y,u);
view(3)
axis([min(x) max(x) min(y) max(y) -1 1])
xlabel('x')
ylabel('y')

% ****************************************************************

hwb = waitbar(0,'Initializing waitbar...');
for n=1:nSteps

    perc = round(100*(n-1)/nSteps);
    waitbar(perc/100,hwb,sprintf('%d%% along...',perc))

    t = t + 0.5*dt;
    um1 = u;
    if mod(n,2)==0,sweep = 'X'; else sweep = 'Y'; end;
    u = alternatePRsteps(u,um1,MX,MY,t,dt,x,y,xc,yc,side_x,side_y,delt_x,delt_y,alpha,↙
sourceFun,bcType,bcFun,sweep);
    u = ghostCells(u,bcType,bcFun,t,xc,yc,delt_x,delt_y,alpha);

    t=t+0.5*dt;
    um1 = u;
    if mod(n,2)==0,sweep = 'Y'; else sweep = 'X'; end;
    u = alternatePRsteps(u,um1,MX,MY,t,dt,x,y,xc,yc,side_x,side_y,delt_x,delt_y,alpha,↙
sourceFun,bcType,bcFun,sweep);
    u = ghostCells(u,bcType,bcFun,t,xc,yc,delt_x,delt_y,alpha);

    if mod(n,reportStep)==0
        figure(hf);
        clf(gcf);
        subplot(1,3,1)
        h=surf(X,Y,u);
        view(3)
        colorbar
        axis([min(x) max(x) min(y) max(y) -1 1])
        xlabel('x')
        ylabel('y')
        subplot(1,3,2)
        h=surf(X,Y,ua(X,Y,t));
        view(3)
        colorbar
        axis([min(x) max(x) min(y) max(y) -1 1])
        xlabel('x')
        ylabel('y')
        subplot(1,3,3)
        uarange = max(max(ua(X(2:end-1,2:end-1),Y(2:end-1,2:end-1),t)))-min(min(ua(X(2:↙
end-1,2:end-1),Y(2:end-1,2:end-1),t)));
        h=surf(X(2:end-1,2:end-1),Y(2:end-1,2:end-1),(100/uarange)*abs(u(2:end-1,2:end-↙
1)-ua(X(2:end-1,2:end-1),Y(2:end-1,2:end-1),t)));
        title(['Maximum absolute error: ',num2str(max(max((100/uarange)*abs(u(2:end-↙
1,2:end-1)-ua(X(2:end-1,2:end-1),Y(2:end-1,2:end-1),t)))),2),'%'])
        view(3)
        colorbar
```

```
        axis([min(x) max(x) min(y) max(y) 0 100])
        xlabel('x')
        ylabel('y')
    end;

end;
close(hwb)

return;

end

function u = alternatePRsteps(u,um1,MX,MY,t,dt,x,y,xc,yc,side_x,side_y,delt_x,delt_y,↙
alpha,sourceFun,bcType,bcFun,sweep)

switch sweep
    case{'x','X'}
        for j=2:MY+1

            source =  sourceFun(xc,yc(j-1)*ones(size(xc)),t-0.25*dt,alpha);
            dFlux = diffusiveFlux(um1,side_x,delt_y,2:MX+1,j-1:j,alpha);
            source = source + (dFlux(:,2)-dFlux(:,1))./(side_x.*side_y(j-1));
            u(2:end-1,j) = diffusion1d(um1(:,j),x,side_x,side_y(j-1),delt_x,dt,alpha,↙
source,{bcType.west,bcType.east},[bcFun.west(t,yc(j-1),alpha),bcFun.east(t,yc(j-1),↙
alpha)]);

        end;
    case{'y','Y'}
        for i=2:MX+1

            source =  sourceFun(xc(i-1)*ones(size(yc)),yc,t-0.25*dt,alpha);
            dFlux = diffusiveFlux(um1',side_y,delt_x,2:MY+1,i-1:i,alpha);
            source = source + (dFlux(:,2)-dFlux(:,1))./(side_y.*side_x(i-1));
            u(i,2:end-1) = diffusion1d(um1(i,:)',y,side_y,side_x(i-1),delt_y,dt,alpha,↙
source,{bcType.south,bcType.north},[bcFun.south(t,xc(i-1),alpha),bcFun.north(t,xc(i-1),↙
alpha)]);

        end;
end;

return;

end

function [x,xc,side_x,delt_x] = meshGen(HX,MX,meshType)

switch meshType
    case{'uniform'}
        side_x = (HX/MX)*ones(MX,1);
        x = 0:side_x(1):HX; x = x(:);
        xc = 0.5*(x(2:end)+x(1:end-1));
        delt_x(MX+1) = side_x(MX);
        delt_x(2:MX) = xc(2:end)-xc(1:end-1);
        delt_x(1) = side_x(1);
        delt_x = delt_x(:);
```

```
end;

return;

end

function u = ghostCells(u,bcType,bcFun,t,xc,yc,delt_x,delt_y,alpha)

switch bcType.west
    case{'D'}
        u(1,2:end-1) = -u(2,2:end-1) + 2.0*bcFun.west(t,yc,alpha)';
    case{'N'}
        u(1,2:end-1) = u(2,2:end-1) + bcFun.west(t,yc,alpha)'.*(delt_x(1)/alpha);
end;

switch bcType.east
    case{'D'}
        u(end,2:end-1) = -u(end-1,2:end-1) + 2.0*bcFun.east(t,yc,alpha)';
    case{'N'}
        u(end,2:end-1) = u(end-1,2:end-1) + bcFun.east(t,yc,alpha)'.*(delt_x(end)↙
/alpha);
end;

switch bcType.south
    case{'D'}
        u(2:end-1,1) = -u(2:end-1,2) + 2.0*bcFun.south(t,xc,alpha);
    case{'N'}
       u(2:end-1,1) = u(2:end-1,2) + bcFun.south(t,xc,alpha).*(delt_y(1)/alpha);
end;

switch bcType.north
    case{'D'}
        u(2:end-1,end) = -u(2:end-1,end-1) + 2.0*bcFun.north(t,xc,alpha);
    case{'N'}
       u(2:end-1,end) = u(2:end-1,end-1) + bcFun.north(t,xc,alpha).*(delt_y(end)↙
/alpha);
end;

return;

end


function dFlux = diffusiveFlux(u,side_x,delt_y,IX,IY,alpha)

dFlux = zeros(length(IX),length(IY));
for i=1:length(IX)
    ii = IX(i);
    for j=1:length(IY)
        jj = IY(j);
        dFlux(i,j) = alpha*(u(ii,jj+1)-u(ii,jj))*side_x(ii-1)/delt_y(jj);
    end;
end;
```

```
return;

end


% Solve 1D-diffusion problem by finite-volume method
function u = diffusion1d(um1,x,side_x,side_y,delt_x,dt,alpha,source,bcType,bcFun)

N = length(side_x);

aw = -(0.5*dt*alpha*side_y)./delt_x(1:end-1);
ae = -(0.5*dt*alpha*side_y)./delt_x(2:end);
ap = side_y*side_x - (ae + aw);
r  = um1(2:end-1).*(side_x.*side_y) + (0.5*dt)*(side_x.*side_y).*source;

% Enforce b-conds
[aw,ap,ae,r] = boundaryConditions(aw,ap,ae,r,bcType,bcFun,delt_x(1),delt_x(end),alpha);

% Could be prefactorized whenever "alpha" is independent of "u" and dt is
% fixed.
A = spdiags([[aw(2:end);0] ap [0;ae(1:end-1)]],-1:1,N,N);

u  = A\r;

return;

end

function [bcType,bcFun] = defineBoundaryConditions

bcType.west = 'D';
bcType.east = 'D';
bcType.north = 'N';
bcType.south = 'N';

bcFun.west  = @(t,y,alpha) (zeros(size(y)));
bcFun.east  = @(t,y,alpha) (zeros(size(y)));
bcFun.south = @(t,x,alpha) (-alpha*x.*sin(2*pi*x).*exp(-t^2));
bcFun.north = @(t,x,alpha) ( alpha*x.*sin(2*pi*x).*exp(-t^2));

return;

end

function [aw,ap,ae,r] = boundaryConditions(aw,ap,ae,r,bcType,bcFun,delt_xw,delt_xe,↙
alpha)

switch bcType{1}
    case{'D'}
        ap(1) = ap(1)-aw(1);
        r(1)  = r(1)-2.0*aw(1)*bcFun(1);
    case{'N'}
        ap(1) = ap(1)+aw(1);
        r(1)  = r(1) - aw(1)*bcFun(1)*delt_xw/alpha;
```

```
end;

switch bcType{2}
    case{'D'}
        ap(end) = ap(end)-ae(end);
        r(end)  = r(end)-2.0*ae(end)*bcFun(2);
    case{'N'}
        ap(end) = ap(end)+ae(end);
        r(end)  = r(end) - ae(end)*bcFun(2)*delt_xe/alpha;
end;

return;

end
```