**Question 3**

Npart = 1; *//# de particules*

*// Espace*

J = 50 ; dx = 1.0/J ;

xx = linspace(dx,1,J) ; *//pour la graphe*

*// Temps*

T = 1 ; dt = 0.02;

Niter = T/dt ;

*// Les conditions initiales*

*// Pour avoir une densité de probabilité, on enforce une normalization*

*// de la distribution initiale.*

disint = ones(1,J).\*rand(1,J);*//une distribution random*

*//disint = ones(1,J); //une distribution uniforme*

*//disint = 1-cos(2\*%pi\*xx); //la distribution initiale pour question 4*

normalize = sum(disint);

uu0 = disint/normalize \* Npart;

uu = uu0 ;

*// Les probabilités*

*// Ici on fixe les probabilités et calcule les coefficients.*

*// Mais si on a les coefficients, on peut calculer*

*// les probabilités à partir de là.*

pdest = 0.1; *//destruction*

pplus = 0.4; *//la particule va à j+1*

pminus = 0.2; *//la particule va à j-1*

prest = 1-pdest-pplus-pminus; *//la particule reste à j*

alpha = pdest/dt; *//coefficient de destruction*

D = pminus \* dx \* dx / dt; *//coefficient de diffusion*

V = pplus-pminus \* dx / dt; *//coefficient d'advection*

disp ("alpha = ");

disp (alpha);

disp ("D = ");

disp (D);

disp ("V = ");

disp (V);

*//Shift*

iiL = [2:J 1] ; *//shift à gauche*

iiR = [J 1:J-1] ;

*//Pour chaque pas de temps*

for n = 1:Niter

drawlater();

uu = prest\*uu + pplus\*uu(iiR)+pminus\*uu(iiL);

clf;

*// La distribution initiales en rouge*

plot2d(xx,uu0,rect=[0,0,1,Npart/10], style = 5) ;

*// La simulation en noir*

plot2d(xx,uu,rect=[0,0,1,Npart/10]) ;

drawnow();

end

**Question 5**

Npart = 1;

*//espace*

J = 50 ; dx = 1/J ;

xx = linspace(dx,1,J) ; *//pour la graphe*

*//temps*

T = 1 ; dt = 0.0001;

Niter = T/dt ;

*//les conditions initiales*

uu0 = (1-cos(2\*%pi\*xx))\*Npart/2 *//la distribution initiale*

uu = uu0 ;

*// Les coefficients*

alpha = 1; *//coefficient de destruction*

D = 1; *//coefficient de diffusion*

V = 1; *//coefficient d'advection*

disp ("alpha = ");

disp (alpha);

disp ("D = ");

disp (D);

disp ("V = ");

disp (V);

*//Les Probabilités*

pdest = alpha\*dt; *//de destruction*

pminus = D\*dt/dx/dx; *//d'aller à j-1*

pplus = V\*dt/dx+pminus; *//d'aller à j+1*

prest = 1-pdest-pplus-pminus; *//de rester à j*

disp ("P(destruction) = ");

disp (pdest);

disp ("P(j-1) = ");

disp (pminus);

disp ("P(j+1) = ");

disp (pplus);

disp ("P(j) = ");

disp (prest);

if (prest>=0) then *//Si la condition CFL est satisfait*

*//Shifters*

iiL = [2:J 1] ;

iiR = [J 1:J-1] ;

*//chaque pas du temps*

for n = 1:Niter;

uu = prest\*uu + pplus\*uu(iiR)+pminus\*uu(iiL);

*//La solution explicite*

sol = (1-cos(2\*%pi\*(V\*n\*dt-xx))\*exp(-4\*%pi^2\*D\*n\*dt))\*exp(-alpha\*n\*dt)\*Npart/2; *//n\*dt=t est un pas du temps*

erreur=sol-uu;

differencelinf(1,n) = norm(erreur,'inf');

differencel2(1,n)=norm(erreur,2);

*//On trace la graphe pour chaque 10eme pas*

if (modulo(n,10)==0)

drawlater() ;

clf ;

subplot(2,2,1);

plot2d(xx,uu,rect=[0,0,1,Npart]) ;

xtitle( 'Simulation', 'Espace', 'Probabilité' ) ;

subplot(2,2,2);

plot2d(xx,sol,rect=[0,0,1,Npart],style=5);

xtitle( 'Solution', 'Espace', 'Probabilité' ) ;

*//Trace d'erreur :*

*// En bleu : l'erreur absolu,*

*// En vert: maximum atteint pour la norme*

subplot(2,2,3);

plot2d(xx,erreur,rect=[0,0,1,0.002\*Npart],style=2);

plot2d(xx,ones(1,J)\*max(differencelinf),rect=[0,0,1,0.002\*Npart],style=3);

xtitle( 'Erreur avec Linf borne', 'Espace', 'Probabilité' ) ;

subplot(2,2,4);

plot2d(xx,erreur,rect=[0,0,1,0.002\*Npart],style=2);

plot2d(xx,ones(1,J)\*max(differencel2),rect=[0,0,1,0.01\*Npart],style=3);

xtitle( 'Erreur avec L2 borne', 'Espace', 'Probabilité' ) ;

drawnow();

end

end

disp ("Max Linf Erreur: ")

disp(max(differencelinf));

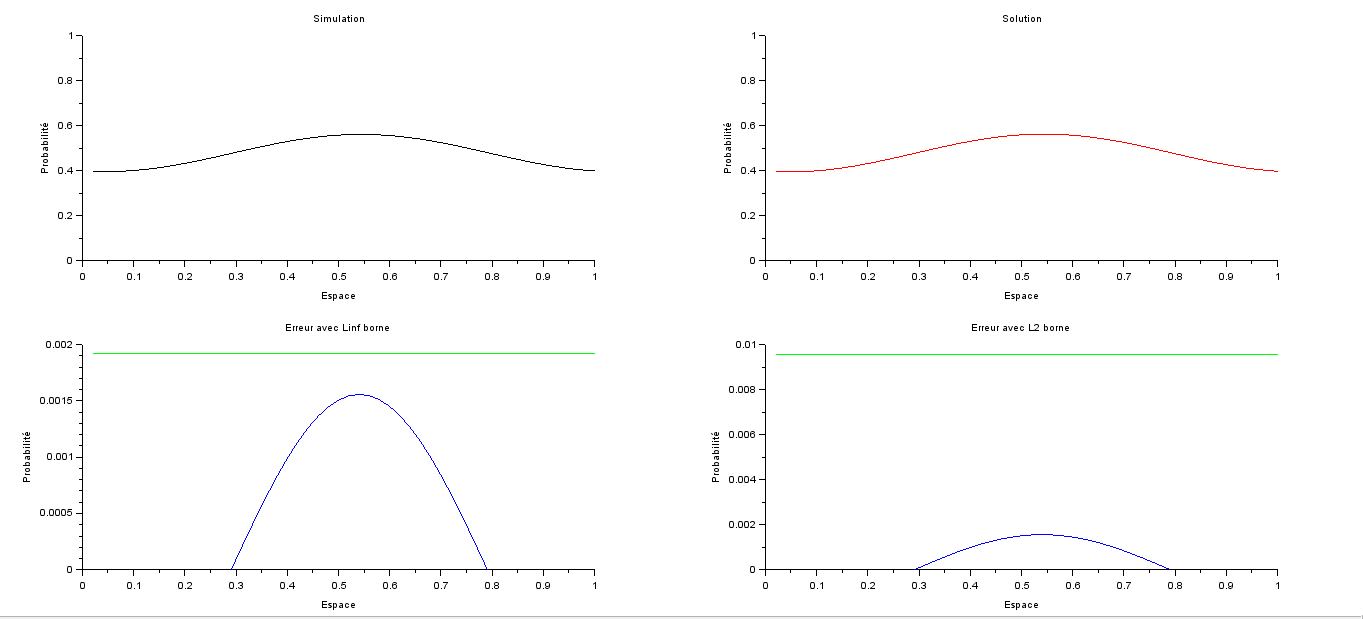
disp ("Max L2 Erreur: ")

disp(max(differencel2));

else

disp("Erreur : P(j)<0, condition CFL pas satisfait!")

end

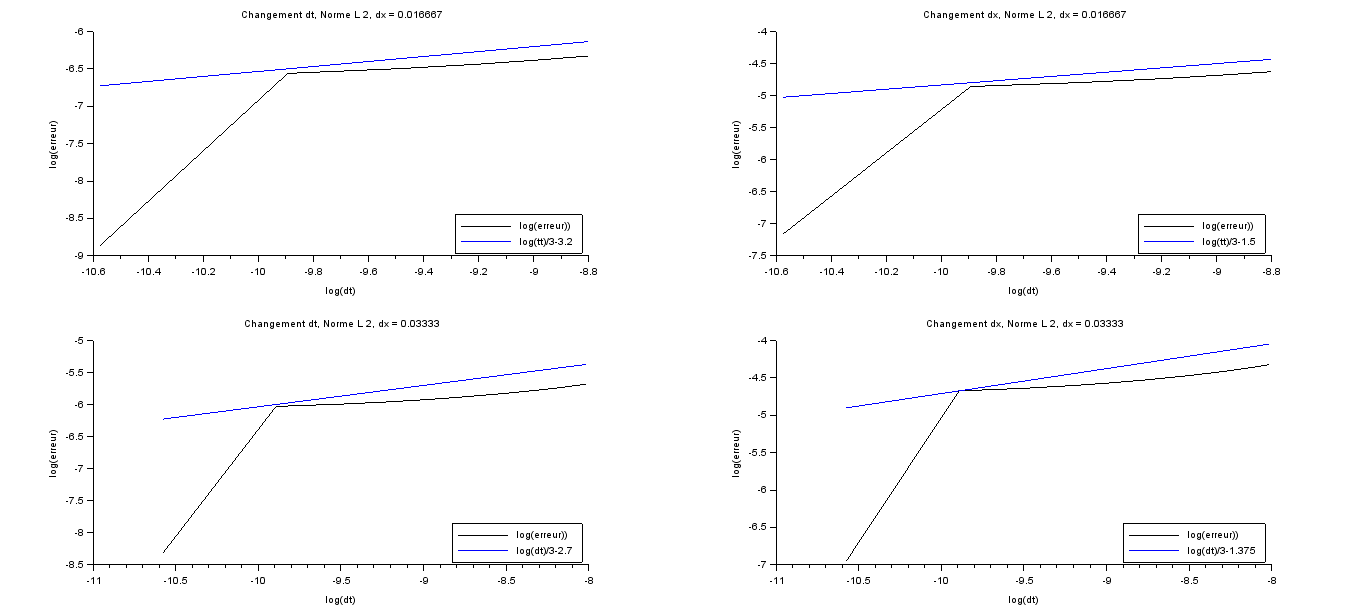
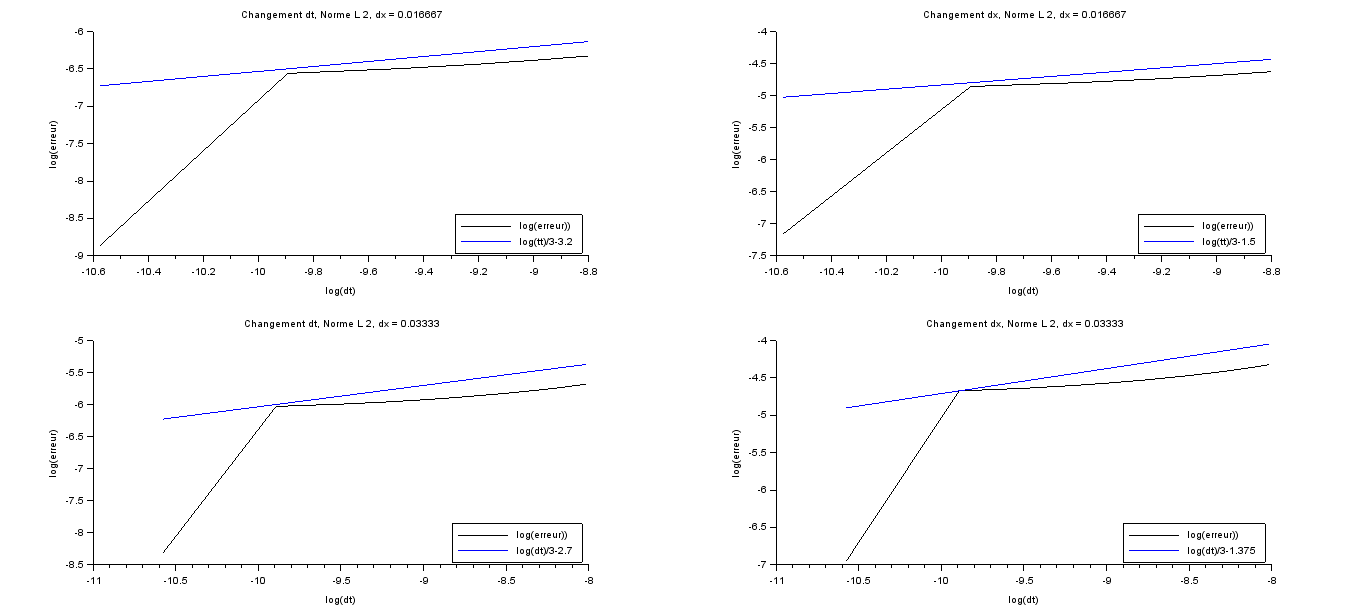


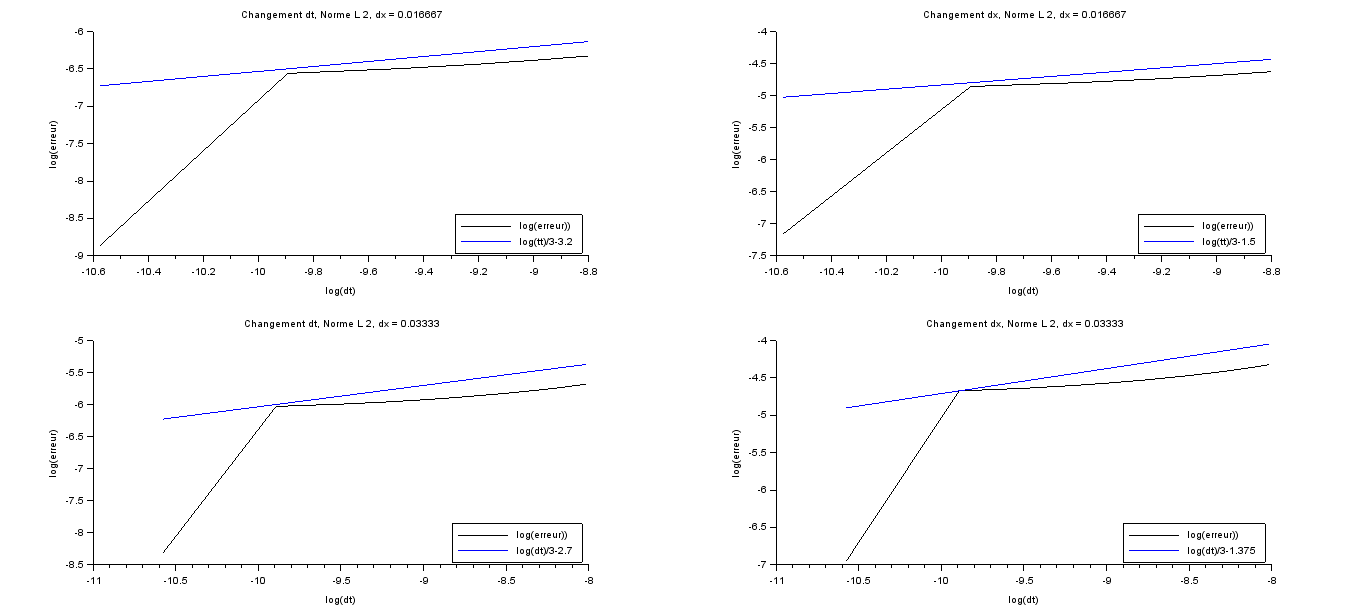
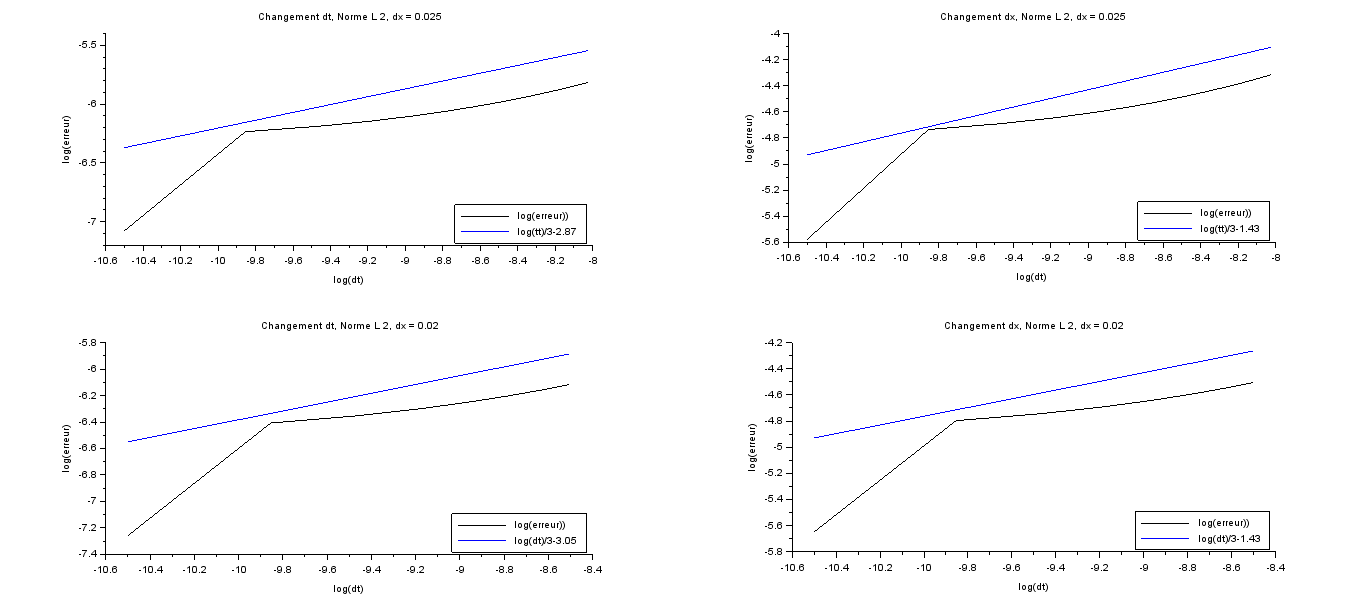
La graphe produit par la simulation

Pour question 5, on a choisi de ne pas normaliser la distribution initiale pour avoir une densité de probabilité, vu que cela change la solution explicite.

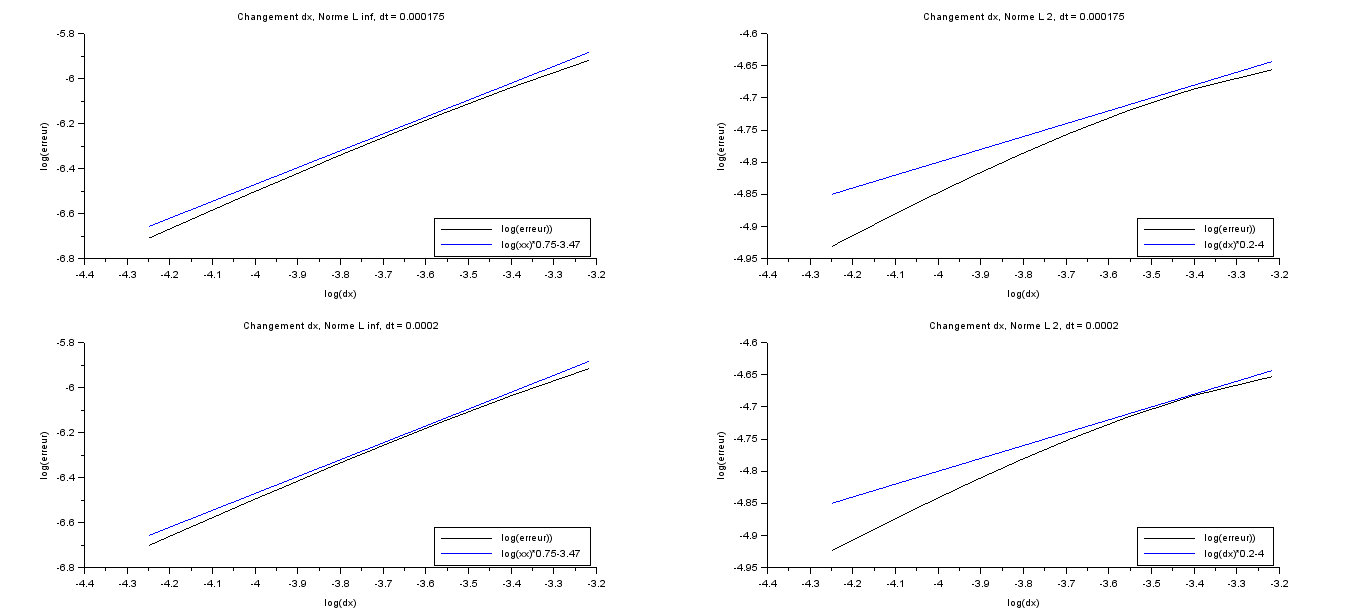
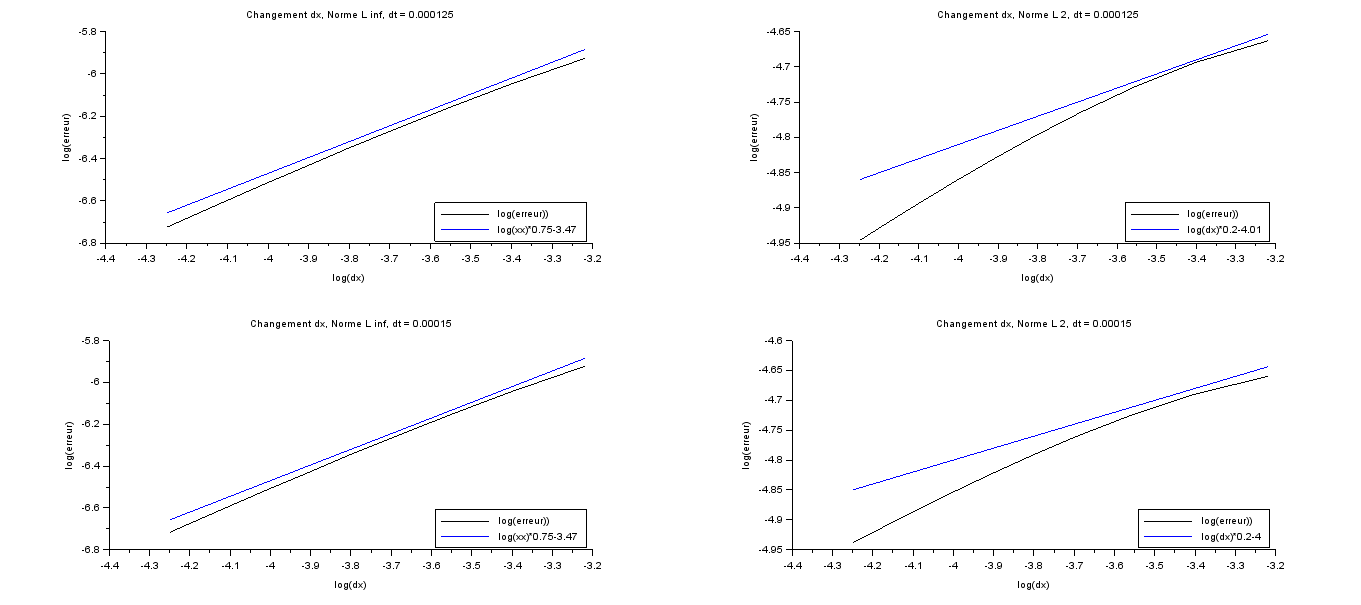
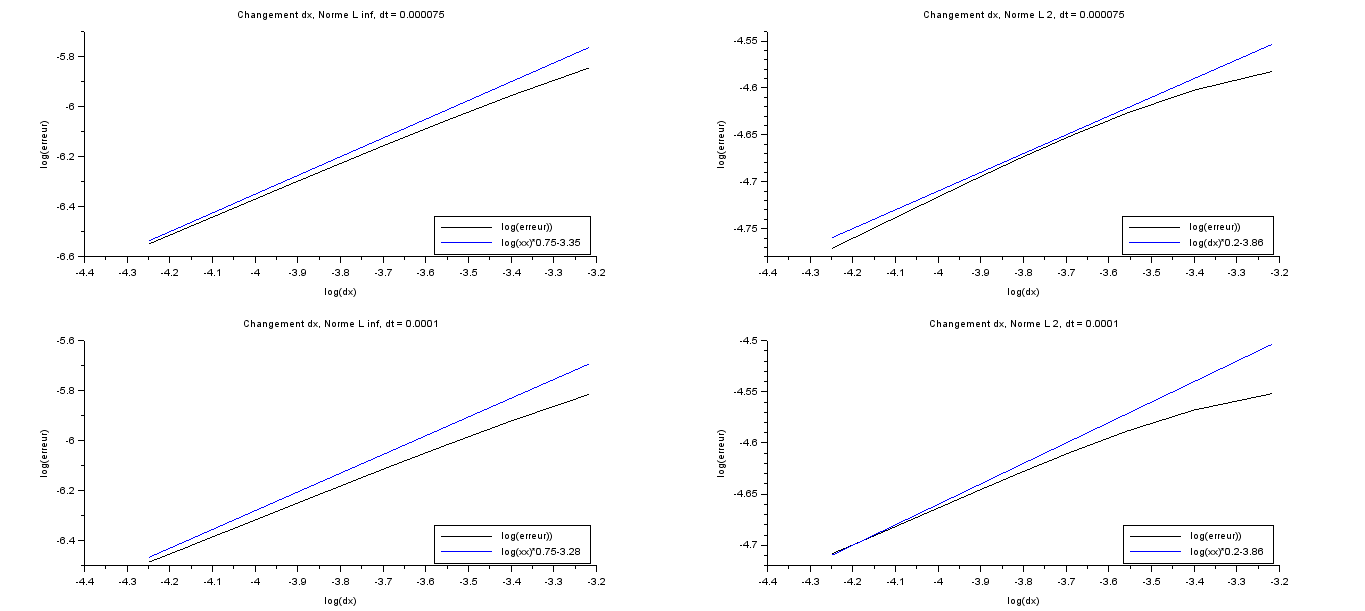
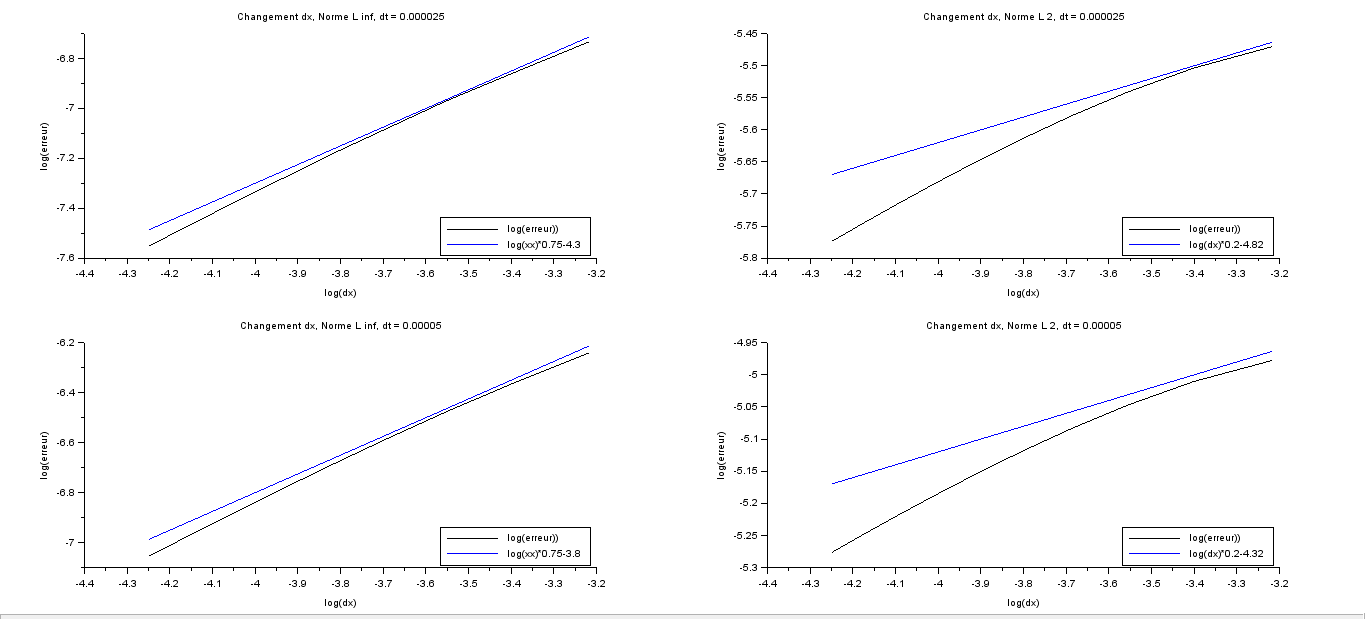
Pour l’analyse d’erreur, on prend la différence entre la solution et la simulation et considère les et normes. On veut montrer que l’erreur est , c’est à dire .

On trace alors le logarithme de l’erreur contre le logarithme de (et ou respectivement). On a donc (ou, respectivement, et ). En considérant la pente de la graphe tracée, on le majore pour trouver , , et .

On varie d’abord dt en tenant dx constante, pour dx = 1/J et J = 30 à 60 par incréments de 10. 



Ici, on trouve que la pente est majorée par un coefficient de 1/3 pour dt, pour les deux normes.

On fait pareil pour dx en tenant dt constante, pour dt = 0.000025 à dt = 0.0002 par incréments de 0.000025.

On majore l’erreur par un coefficient de ¾ pour et 1/5 pour la norme, pour dx.

**Question 6**

On a exactement le même code que question 5, sauf que dans le boucle « for n = 1:Niter » on ajoute, après avoir calculé uu :

*//conditions aux bords*

uu(1,1) = 0;

uu(J) = uu(J-1);

La première condition du bord gauche est . Cela corresponde à un thermostat à température constant et nul à . L’autre condition de bord est . La flux sortant du bord droit est donc nul et signifie que le système est thermiquement isolé de l’extérieur à .

N’ayant pas la solution exacte, on fait une analyse d’erreur de troncation qui nous amène à la convergence et stabilité du schéma.

L’équation qu’on veut résoudre est le suivant :

Et le schéma qu’on utilise est pareil à celui de question 5 :

La deuxième condition de bord est schématisé par

En assumant que la solution est exacte à , on analyse l’erreur donné par un pas de .

On détail les termes en utilisant les séries de Fourier :

Et on les substitue dans l’erreur de troncation pour trouver :

Mais étant une solution, il vérifie

On a donc que l’erreur de troncation est .