Computer Project 2

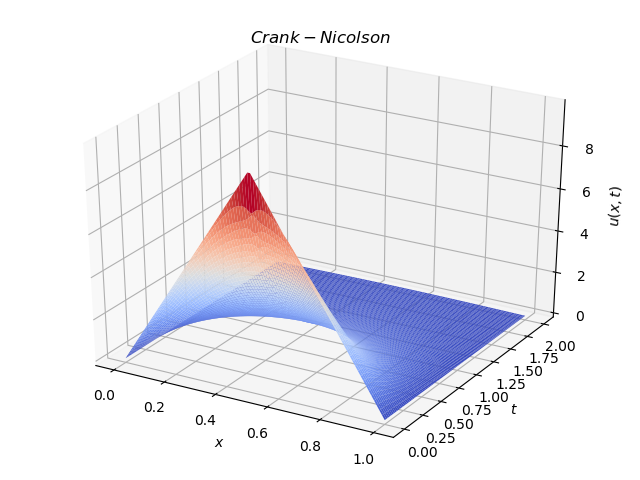
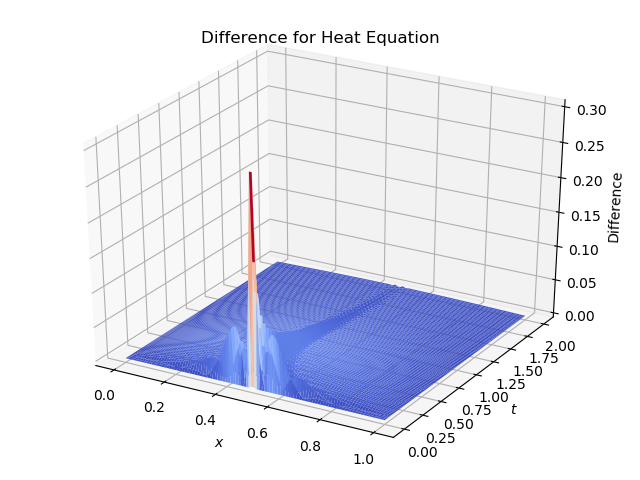
Nicolas Perez

Prof. Sergey Lototsky

4/29/2020

**Problem 1**

**Graphs:**



**Printout of program:**

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits import mplot3d

from scipy import integrate

from math import pi, sin, exp, cos

import sys

def cn\_heat():

# parameters

N = 100 #number of grid points

L = float(1) #size of grid

nsteps = N #number of time steps

dt = 2.0/nsteps #time step

dx = L/(N-1) #grid spacing

r = (0.25 \* dt )/ dx\*\*2

# initialize matrices A, B and b array

A = np.zeros((N-2,N-2))

B = np.zeros((N-2,N-2))

#define matrices A, B

for i in range(N-2):

if i==0:

A[i,:] = [2+2\*r if j==0 else (-r) if j==1 else 0 for j in range(N-2)]

B[i,:] = [2-2\*r if j==0 else r if j==1 else 0 for j in range(N-2)]

elif i==N-3:

A[i,:] = [-r if j==N-4 else 2+2\*r if j==N-3 else 0 for j in range(N-2)]

B[i,:] = [r if j==N-4 else 2-2\*r if j==N-3 else 0 for j in range(N-2)]

else:

A[i,:] = [-r if j==i-1 or j==i+1 else 2+2\*r if j==i else 0 for j in range(N-2)]

B[i,:] = [r if j==i-1 or j==i+1 else 2-2\*r if j==i else 0 for j in range(N-2)]

#initialize grid

x = np.linspace(0,1,N)

t = np.linspace(0,2,nsteps)

#initial conditions (t=0)

u = np.asarray([20\*xx if xx<=0.5 else 20\*(1-xx) for xx in x])

#evaluate right hand side at t=0

rhs = B.dot(u[1:-1])

# solution matrix

sol = np.zeros((nsteps, N))

sol[0, :] = u

# calculate solution for each time step

for j in range(1, nsteps):

sol[j, 1:-1] = np.linalg.solve(A,rhs)

# update rhs

rhs = B.dot(sol[j, 1:-1])

# plotting

fig = plt.figure()

ax = plt.axes(projection='3d')

x, t = np.meshgrid(x, t)

ax.plot\_surface(x, t, sol, rstride=1, cstride=1,

cmap='coolwarm')

ax.set\_title('$Crank-Nicolson$')

ax.set\_xlabel("$x$")

ax.set\_ylabel("$t$")

ax.set\_zlabel("$u(x, t)$")

plt.tight\_layout()

fig.savefig("cn\_heat.png")

return sol

def phi(x):

if x < 0.5:

return 20\*x

else:

return 20\*(1-x)

def fs\_heat():

# parameters

N = 100

max\_iter = 200

# grid

x = np.linspace(0,1, N)

t = np.linspace(0,2, N)

u = np.zeros(shape=(N, N))

# fourier series solution

for i in range(N):

for j in range(N):

for k in range(1, max\_iter):

integral = integrate.quad(lambda x: phi(x)\*sin(pi\*k\*x), 0, 1)[0]

u[i, j] += 2 \* (integral) \* exp(-0.25\*pow(pi\*k, 2)\*t[i]) \* sin(pi\*k\*x[j])

# difference between crank nicolson and fourier series

diff = abs(u-cn\_heat())

# plotting

fig = plt.figure()

ax = plt.axes(projection='3d')

x, t = np.meshgrid(x, t)

ax.plot\_surface(x, t, diff, rstride=1, cstride=1,

cmap='coolwarm')

ax.set\_title('Difference for Heat Equation')

ax.set\_xlabel("$x$")

ax.set\_ylabel("$t$")

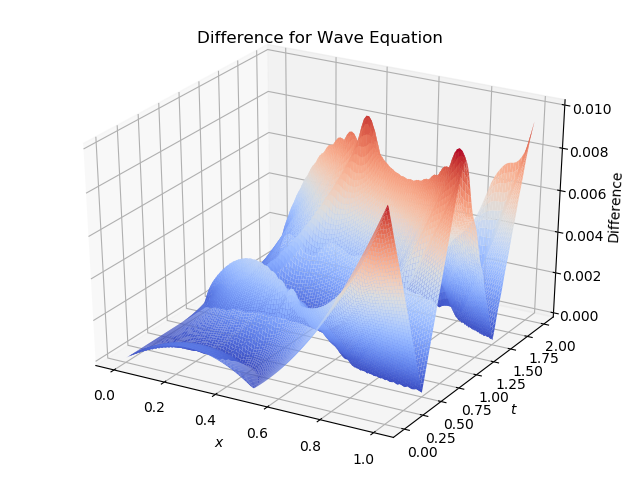
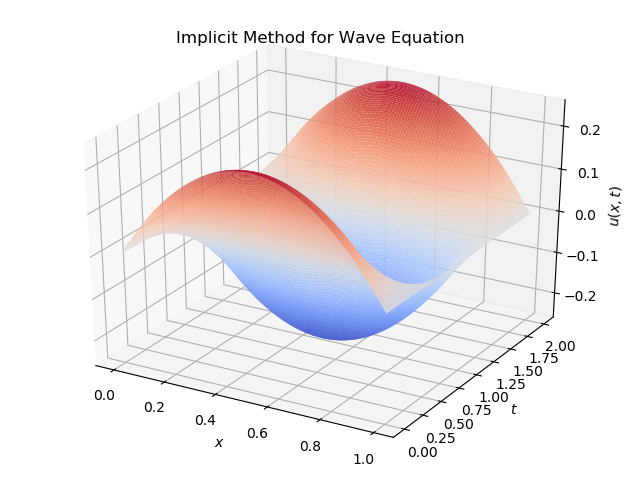
ax.set\_zlabel("Difference")

plt.tight\_layout()

fig.savefig("diff\_heat.png")

**Problem 2**

**Graphs:**

****

**Printout of program:**

def implicit\_wave():

# define parameters

tau = 0.01

h = 0.01

m2 = pow(tau, 2) / pow(h, 2)

f = lambda x: x\*(1-x)

M = int(1/h)

N = int(2/tau)

# define intervals

x = np.linspace(0, 1, M+1)

t = np.linspace(0, 2, N+1)

# define matrix A

A = np.zeros(shape=(M-1, M-1))

for i in range(M-1):

if i == 0:

A[i, 0] = 2\*(1+m2)

A[i, 1] = -m2

elif i == M-2:

A[i, M-3] = -m2

A[i, M-2] = 2\*(1+m2)

else:

A[i, i-1] = -m2

A[i, i] = 2\*(1+m2)

A[i, i+1] = -m2

# define u with boundary conditions

u = np.zeros(shape=(M+1, N+1))

# define initial conditions

u[1:-1, 0] = [f(i) for i in x[1:-1]]

utt = np.asarray(

[(f(i\*h)-2\*f((i-1)\*h)+f((i-2)\*h))/pow(h,2) for i in range(M-1)])

u[1:-1, 1] = u[1:-1, 0] + (pow(tau, 2)/2)\*utt

# compute rhs for first iteration

rhs = 4\*u[1:-1, 1] - A.dot(u[1:-1, 0])

# solve

for j in range(1, N):

rhs = 4\*u[1:-1, j] - A.dot(u[1:-1, j-1])

u[1:-1, j+1] = np.linalg.solve(A, rhs)

# plotting

fig = plt.figure()

ax = plt.axes(projection='3d')

x, t = np.meshgrid(x, t)

ax.plot\_surface(x, t, u.transpose(), rstride=1, cstride=1,

cmap='coolwarm')

ax.set\_title('Implicit Method for Wave Equation')

ax.set\_xlabel("$x$")

ax.set\_ylabel("$t$")

ax.set\_zlabel("$u(x, t)$")

plt.tight\_layout()

fig.savefig("implicit\_wave.png")

return u[:-1, :-1]

def fs\_wave():

# parameters

M = 100

N = 200

max\_iter = 100

# domain and solution

x = np.linspace(0,1, M)

t = np.linspace(0,2, N)

u = np.zeros(shape=(M, N))

# Fourier Series solution

Ak = lambda k: 2\*integrate.quad(lambda x: x\*(1-x)\*sin(pi\*k\*x), 0, 1)[0]

for i in range(M):

for j in range(N):

for k in range(1, max\_iter):

u[i, j] += Ak(k)\*cos(pi\*k\*t[j])\*sin(pi\*k\*x[i])

# get implicit solution

impl = implicit\_wave()

# difference

diff = abs(u-impl)

# plotting difference

fig = plt.figure()

ax = plt.axes(projection='3d')

x, t = np.meshgrid(x, t)

ax.plot\_surface(x, t, diff.transpose(), rstride=1, cstride=1,

cmap='coolwarm')

ax.set\_title('Difference for Wave Equation')

ax.set\_xlabel("$x$")

ax.set\_ylabel("$t$")

ax.set\_zlabel("Difference")

plt.tight\_layout()

fig.savefig("diff\_wave.png")

# plotting fourier series solution

fig = plt.figure()

ax = plt.axes(projection='3d')

ax.plot\_surface(x, t, u.transpose(), rstride=1, cstride=1,

cmap='coolwarm')

ax.set\_title('Wave Equation Fourier Series')

ax.set\_xlabel("$x$")

ax.set\_ylabel("$t$")

ax.set\_zlabel("$u(x,t)$")

plt.tight\_layout()

fig.savefig("fs\_wave.png")