HW4 Report for Math/CS 471

Xiaomeng Li

10/28/2017

Abstract

This is the HW4 Report. First, the report will give a brief introduction of Math background. Second, the report will outline the procedure and methods I used. Third, I will introduce the data results and try to analyze and discuss them. In the end, I will give a conclusion about this Math problem. This time I use Microsoft Word to write this report.

Introduction

In a logically rectangular domain, I will compute derivatives and integrals. The main logic is that I can treat this rectangular domain as a reference domain for my problem, denoted by Ω_R. $\Omega_R = \{(r,s) \in [-1,1]^2\}$. Using a mapping function $(x,y) = (x(r,s), y(r,s))$, I can map any function $u(x,y) = f(x,y) = f\big(x(r,s), y(r,s)\big) = F(r,s)$. This method allows me to compute derivatives and integrals from the reference domain to get the results of real domain, i.e. from Ω_R to Ω.

A. Differentiation on Ω

$u = u(x,y)$ is a continuously differentiable function of x and y, where $(x,y) \in \Omega$. In order to approximate the partial derivative of u with respect to x and y, I will first write functions $x = x(r,s)$ and $y = y(r,s)$ as functions of r and s, where $\{(r,s) \in [-1,1]^2\}$. Then I can use the chain rule and come up with the following equation:

$$u_x = u_r\, r_x + u_s\, s_x$$
$$u_y = u_r\, r_y + u_s\, s_y$$

Then the calculation is transferred to reference domain. With the Cartesian grid on the reference element $\Omega_R = [-1,1]^2$ and use the standard finite difference formulas, I can get u_r as well as u_s. It is easy to use standard finite difference formulas to get x_r, y_r, x_s, y_s with the known mapping results x_c and y_c. Therefore, I can have r_x, s_x, r_y, s_y with the following equation.

$$\begin{bmatrix} r_x & s_x \\ r_y & s_y \end{bmatrix}\begin{bmatrix} x_r & y_r \\ x_s & y_s \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\text{Therefore, } \begin{bmatrix} r_x & s_x \\ r_y & s_y \end{bmatrix} = \begin{bmatrix} x_r & y_r \\ x_s & y_s \end{bmatrix}^{-1}$$

B. Integration on Ω

It is accessible to use this reference domain to compute integrals with the equation below:

$$\int_\Omega f(x,y)dxdy = \int_{-1}^{1}\int_{-1}^{1} f\big(x(r,s), y(r,s)\big)J(r,s)drds$$

Where $\mathrm{J(r,s)} = x_r y_s - x_s y_r$ is the surface element

Since the domain relates both r and s to the computation process, it is required to use 2D trapezoidal rule. Similar to 1D trapezoidal rule, 2D will require the computation on the four corners, four boundary lines and the parts in the center. Note that before implement the 2D trapezoidal rule, it is necessary to square every element in the integral and use square root and them to keep everything positive.

Assuming: $\iint F(r,s)\, drds$ on $[-1,1]^2$

$$\mathrm{I} = h_r \times h_s \times \left(\tfrac{1}{4}\textstyle\sum f(corners) + \tfrac{1}{2}\sum f(bounday) + f(interior)\right)$$

Procedure and Methods

1. Make and run the program for different functions and mappings. I use three different functions and three different mappings. All the implementations are included in Makefile.

2. Use the equations above to find r_x, s_x, r_y, s_y. After Cartesian grid and mapping, firstly I use r and s to find $u(x,y) = f(x,y) = f\big(x(r,s), y(r,s)\big) = F(r,s)$. Secondly, with the subroutine *differentiate.f90* I use $u(r,s)$ to compute $u_r(r,s)$ and $u_s(r,s)$.
3. Next, it is left to find x_r, y_r, x_s, y_s. Similar to the process above, I use subroutine *differentiate.f90* as well as mapping results x_c and y_c to compute $x_r(r,s)$, $y_r(r,s)$ in r direction; $x_s(r,s)$, $y_s(r,s)$ in s direction.

4. Use the inverse equation below:
$$\begin{bmatrix} x_r & y_r \\ x_s & y_s \end{bmatrix}^{-1} = \frac{1}{x_r \times y_s - y_r \times x_s} \times \begin{bmatrix} y_s & -y_r \\ -x_s & x_r \end{bmatrix} = \begin{bmatrix} r_x & s_x \\ r_y & s_y \end{bmatrix}$$

I can get r_x, s_x, r_y, s_y. It is worth mentioning that during the computation, $x_r \times y_s - y_r \times x_s$ is actually the $\mathrm{J(r,s)}$ which will be used later for 2D trapezoidal rule, therefore I store all the elements in $\mathrm{J(r,s)}$ separately. The u_x and u_y are computed by the following two equations:
$$u_x = u_r\, r_x + u_s\, s_x$$
$$u_y = u_r\, r_y + u_s\, s_y$$

5. I write a subroutine called *approerror.f90* to approximate the error. The error equation e_2 is listed below. Before my implementation, I computed the $(u_{exact})_x$ and $(u_{exact})_y$ with my own calculation on paper and bring the results in Fortran.
$$e_2(h_{r,}h_s) = (\int_\Omega (u_x(x,y) + u_y(x,y) - [(u_{exact})_x + (u_{exact})_y])^2 dxdy)^{1/2}$$

6. In *approerror.f90,* I calculate firstly $e_2 = u_x + u_y - (u_{exact})_x - (u_{exact})_y$. Then I multiply e_2 with $\mathrm{J(r,s)}$ since $\mathrm{J(r,s)} = x_r y_s - x_s y_r$ are all commonly used during this homework. Then I take the square of e_2 and take the square root of e_2 to keep everything positive. In the end, I use the 2D trapezoidal rule to get the integral of e_2.

7. Now it is left to find $\Delta u = u_{xx} + u_{yy}$. I will need to discretize it into
$$u_x = u_r\, r_x + u_s\, s_x$$
$$u_{xx} = (u_x)_r\, r_x + (u_x)_s\, s_x$$

$$u_y = u_r\, r_y + u_s\, s_y$$
$$u_{yy} = (u_y)_r r_y + (u_y)_s\, s_y$$

With the equations above, I will begin to implement all the similar procedures that I used before. I use subroutine *differentiate.f90* and u_x , u_y to find $(u_x)_r$ and $(u_y)_r$ in r direction; $(u_x)_s$ and $(u_y)_s$ in s direction. Then loop over ns and nr to get u_{xx} and u_{yy}. Finally $\Delta u = u_{xx} + u_{yy}$.

Results and Discussion

1. Three functions I used in this homework:

Equation a:
$$\mathrm{x = r + s}$$

$$y = r - s$$
$$u = x + y = 2 \times r$$
$$u_x = 1$$
$$u_y = 1$$

Equation b:

$$\mathrm{x} = \sin(\mathrm{r}) + \cos(\mathrm{s})$$
$$y = \sin(s) + \cos(r)$$
$$u = x + y = \sin(\mathrm{r}) + \cos(\mathrm{s}) + \sin(s) + \cos(r)$$
$$u_x = 1$$
$$u_y = 1$$

Equation c:

$$\mathrm{x} = \sin(\mathrm{r}) + \mathrm{s}$$
$$y = \cos(\mathrm{s}) - r$$
$$u = x - y = \sin(\mathrm{r}) + \mathrm{s} - \cos(\mathrm{s}) + r$$
$$u_x = 1$$
$$u_y = -1$$

2. The corresponding Mapping I got:

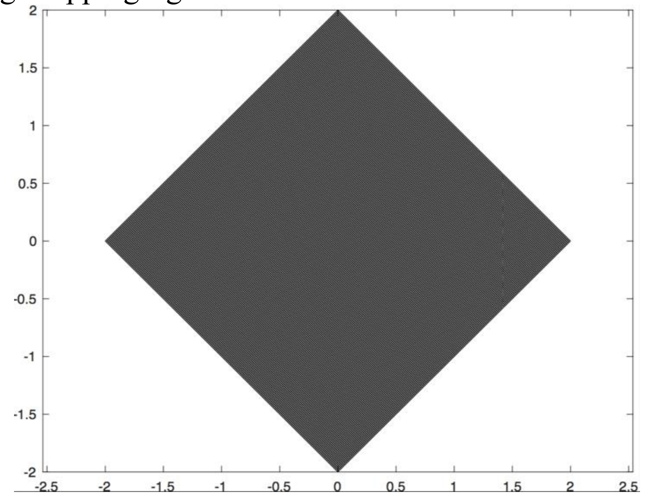

Figure1: The mapping of equation a

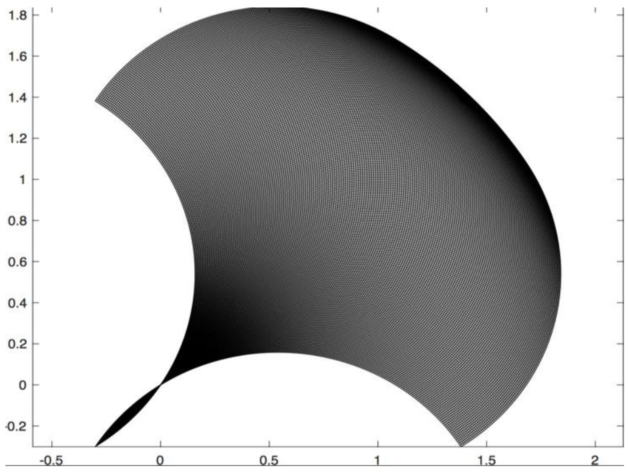

Figure2: The mapping of equation b

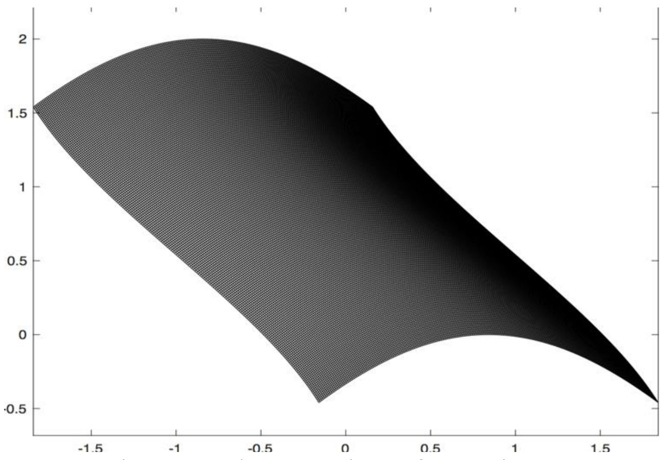

Figure3: The mapping of equation c

The procedure for mapping is easy. Basically I choose the mapping function by myself in *xycoord.f90* and print it by *printdble.f90.* In the end, plot it in MATLAB.

3. Next, I will try to find r_x, s_x, r_y, s_y. I have now the real coordinates of x and y denoted by x_c and y_c. Using subroutine *differentiate.f90* as well as mapping results x_c and y_c to compute $x_r(r,s)$, $y_r(r,s)$ in r direction; $x_s(r,s)$, $y_s(r,s)$ in s direction. At last use inverse matrix to get r_x, s_x, r_y, s_y. Below is an example I have about surface element J with nr = 4, ns = 6 from equation c.

1.5257317558264778	1.7291579749611807	1.8308710845285301	1.7291579749611801	1.5257317558264791	1.3682625109477762	1.5107577082055257
1.5820053068344011	1.5107577082055261	1.3682625109477762	1.1948567669585226	1.2702544863009066	1.3079533459720989	1.2702544863009069
1.1948567669585226	0.99999999999999978	0.99999999999999989	0.99999999999999989	1.0000000000000000	1.0000000000000000	0.80514323304147661
0.72974551369909280	0.69204665402790067	0.72974551369909291	0.80514323304147717	0.63173748905222360	0.48924229179447432	0.41799469316559901
0.48924229179447387	0.63173748905222360	0.47426824417352054	0.27084202503882060	0.16912891547147080	0.27084202503882093	0.47426824417352098

Table 1: Surface element J example when nr = 4, ns = 6 from equation c.

4. Next, I will try to find the area of the mapping result using the reference domain $\{(r,s) \in [-1,1]^2\}$. Note that here 2D trapezoidal rule will be used. The table below shows the results of the equation a, b, and c's mapping areas in real coordinates using reference domain calculation. All of them all come with nr = 30, ns = 60.

Equation	Area
a	8.0602962962963680
b	4.9958060179981167
c	5.3090318595935599

Table 2: The 2D trapezoidal rule results of mapping areas

5. Next I write the subroutine called *approerror.f90* to find the corresponding error with their grid spacing. Note that the plots I have in here use the maximum error and $h = \frac{h_{r+}h_s}{2}$. $error = \max(u_y + u_y - \left[(u_{exact})_x + (u_{exact})_y\right])$. Also, here in order to plot more points, I set nr starting from 10 until 200, ns always equal to corresponding nr + 30. Error is taken after all the elements in $(u_y + u_y - \left[(u_{exact})_x + (u_{exact})_y\right])$ are taken square and square root.

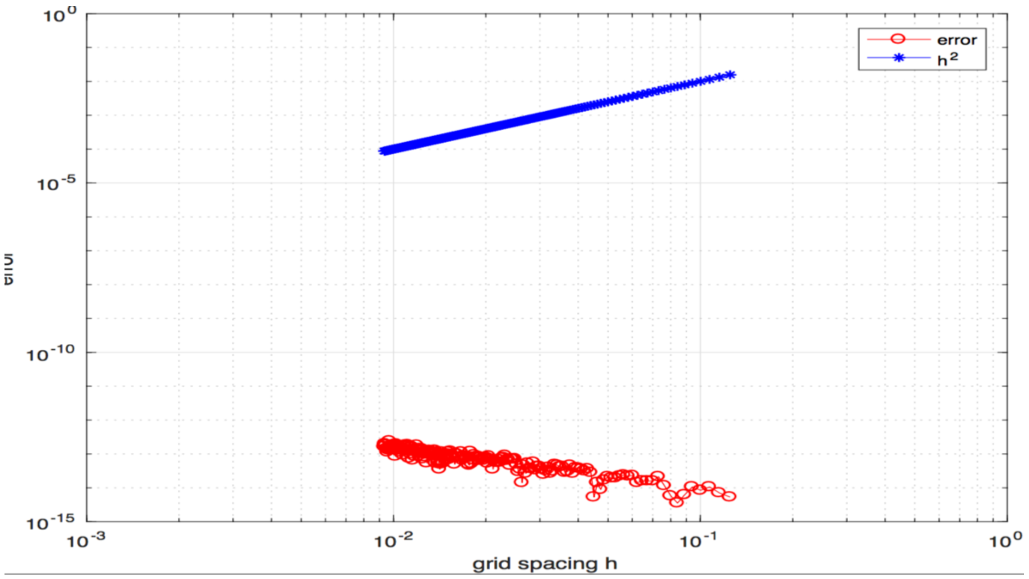

Figure 4: grid spacing vs maximum error for equation a.

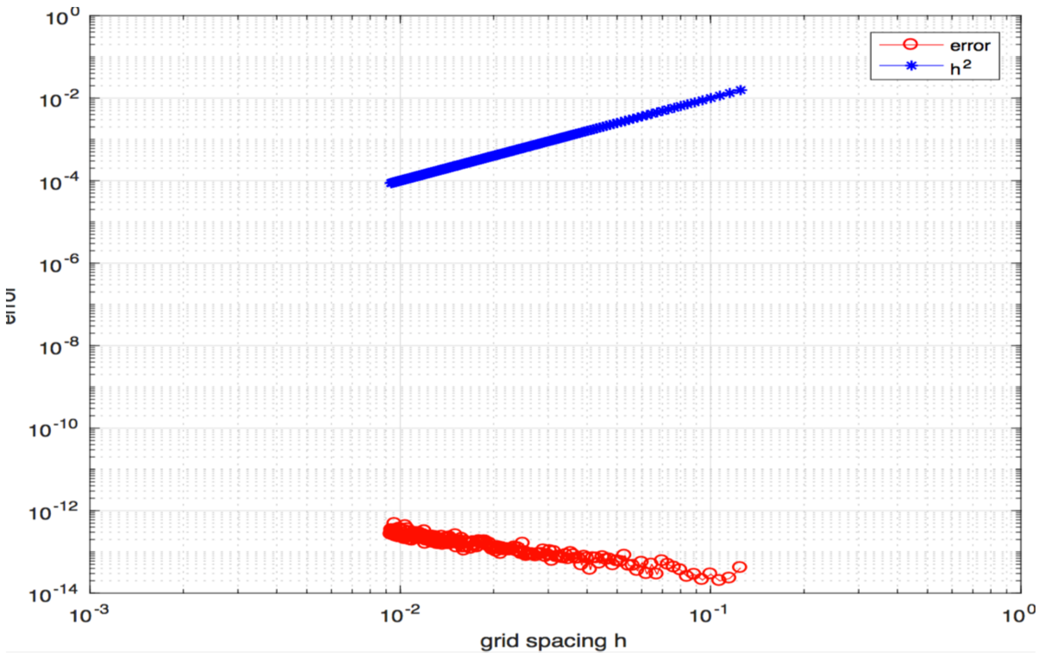

Figure 5: grid spacing vs maximum error for equation b.

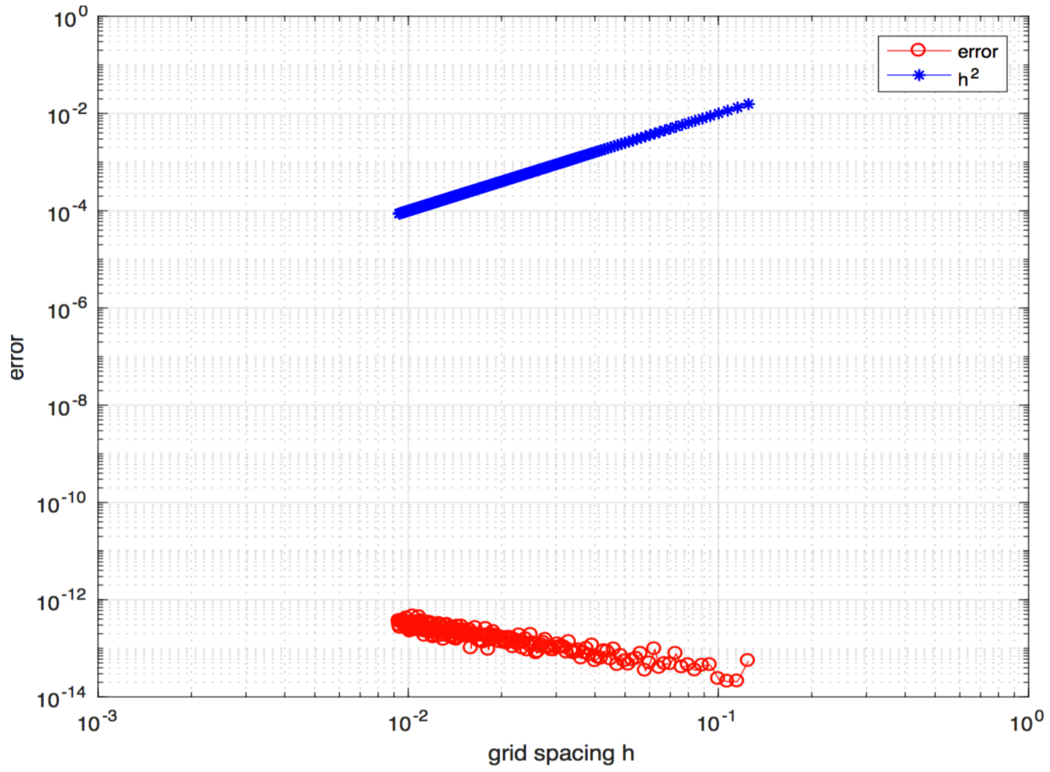

Figure 6: grid spacing vs maximum error for equation c.

Discussion of results:
From the three equations' grid spacing h vs. maximum error plot results outlined above, it is clear that my result does not lead to a convergence. Although I did not get the good results, I can try to analyze how would it be if I did everything right.

First, the error I use is equal to $\max(u_y + u_y - [(u_{exact})_x + (u_{exact})_y])$. $(u_y + u_y - [(u_{exact})_x + (u_{exact})_y])$ will produce a (nr+1) by (ns+1) matrix and I use the maximum value of it. Correspondingly, I also use 2D trapezoidal rule to compute the error outlined below.

$$e_2(h_{r,} h_s) = (\int_\Omega (u_x(x,y) + u_y(x,y) - [(u_{exact})_x + (u_{exact})_y])^2 dxdy)^{1/2}$$

If you want to get the $e_2(h_{r,} h_s)$ results you can try "sum2" in the code since it will produce the double integral e_2 results. The three figures below show the $e_2(h_{r,} h_s)$ I got from my 2D trapezoidal rule. Maybe something is wrong with my trapezoidal rule coding.

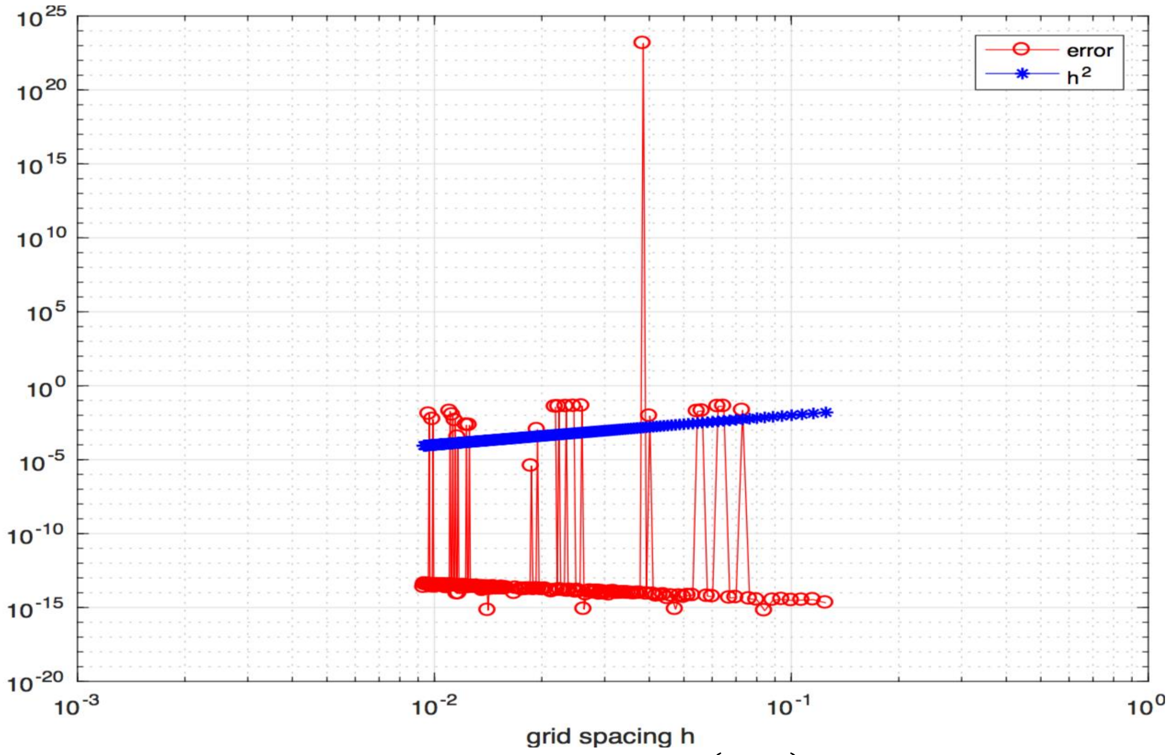

Figure 7: grid spacing vs double integral $e_2(h_{r,} h_s)$ error for equation a

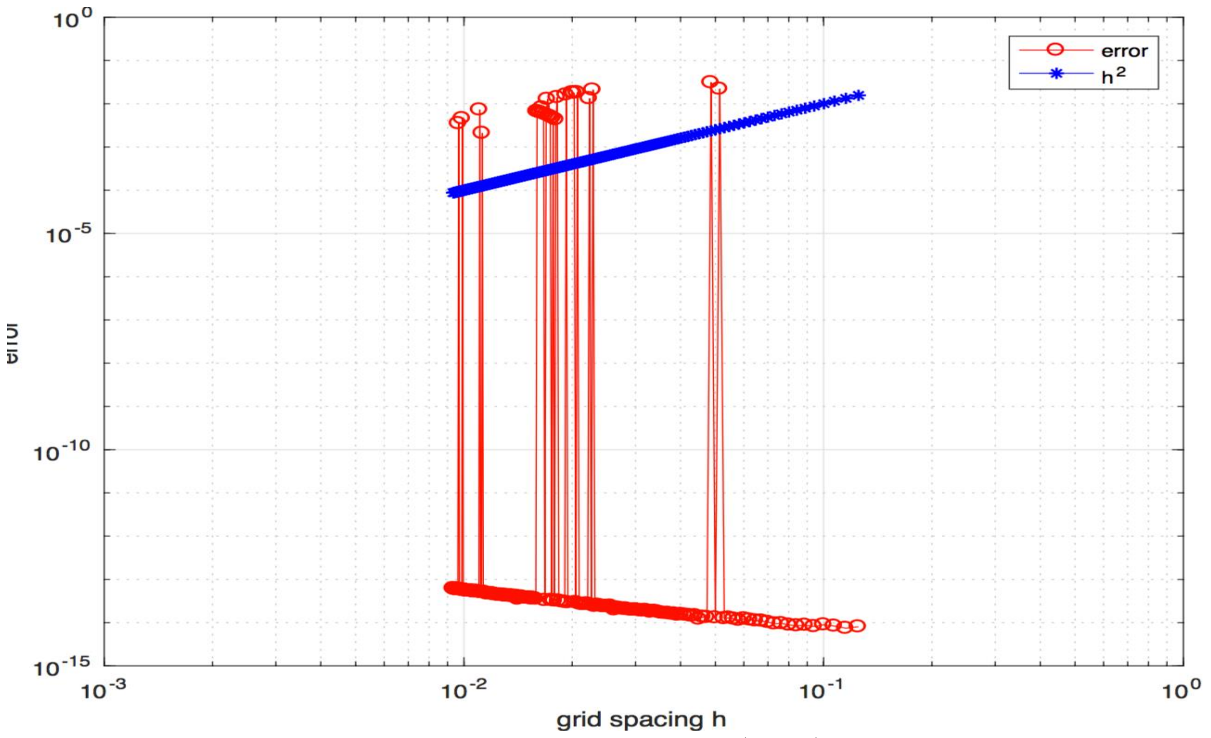

Figure 8: grid spacing vs double integral $e_2(h_{r,}h_s)$ error for equation b

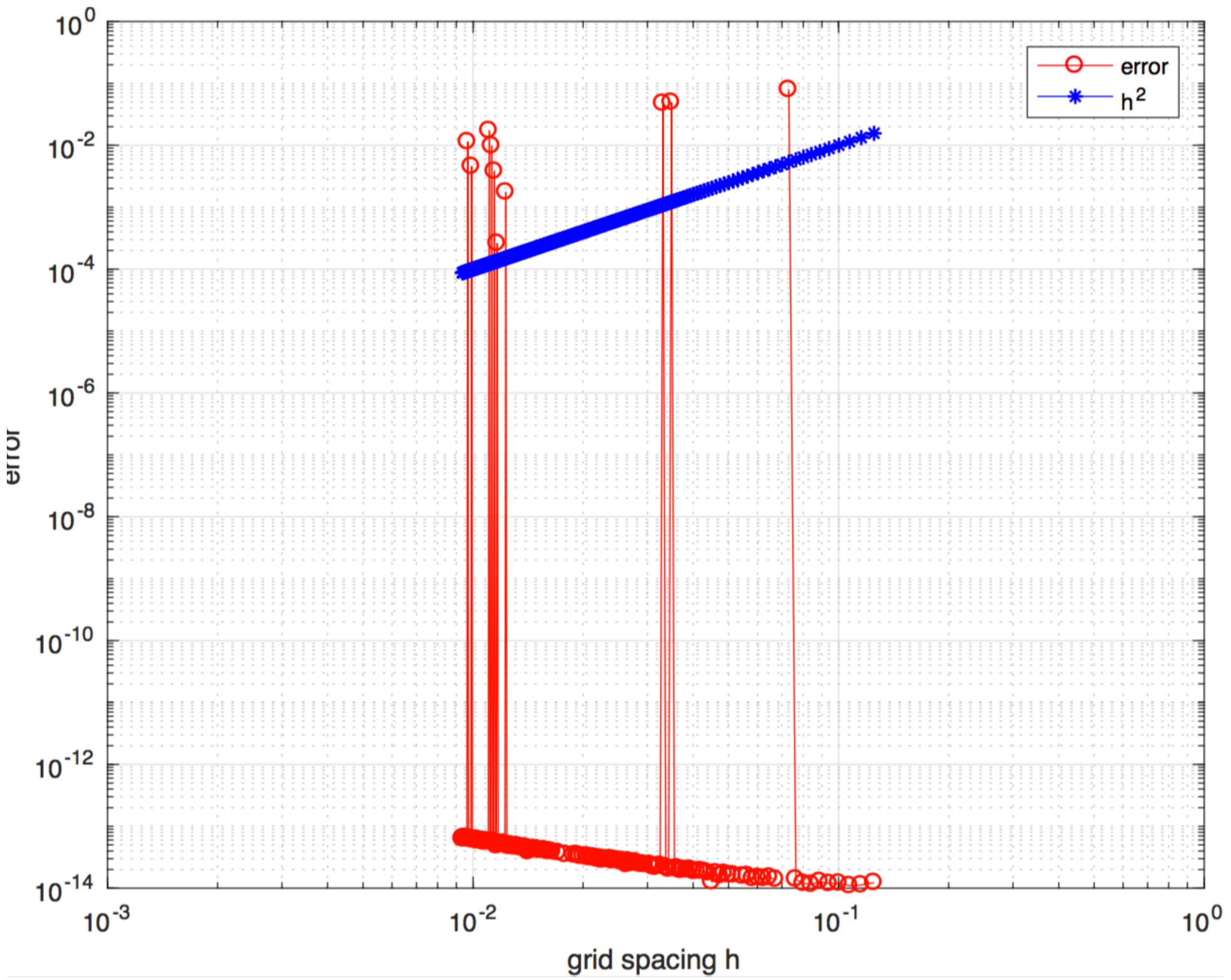

Figure 9: grid spacing vs double integral $e_2(h_{r,}h_s)$ error for equation c

If everything went well, my results were supposed to converge. In this homework, finite difference approximations were used. Here only first-order derivative is considered. However, the second-order, even n.th order derivative is also doable using the ideas in the Procedure and Methods section (7). The first order derivative we used in here is a three point second order accurate stencil on an equidistant grid. In the interior we have a centered stencil -0.5, 0, 0.5. To the left we have a biased stencil -1.5, 2.0, -0.5. To the right we have a biased stencil 1.5, -2.0, 0.5.

The results from the output of the above program is supposed be parallel with the line h^2. If the slope of the error is the same as a h^2, it indicates that the implementation may be correct. Although my plots are definitely wrong, I think the right plot should be parallel with the blue line.

6. In the end, I will try to get $\Delta u = u_{xx} + u_{yy}$. Below is the Δu I got when I used equation c with nr = 6, ns = 4.

1.0395670499947282E-014	1.1413570864597374E-015	4.4045650332931616E-016	6.1022188753494668E-016	5.2344764453507170E-016
-9.6321379801985838E-016	-1.9775490452965456E-015	-3.1072075593506636E-015	-2.5102623168623068E-015	-2.6169622578075741E-016
4.5619493466782342E-016	3.1133682108892607E-016	-3.6571684194224017E-016	-2.6134499302426634E-015	-1.4300373318946854E-014
-3.9968028886505635E-015	-3.2502386389017319E-016	8.1783067480604730E-018	3.4110995088492084E-016	-5.6462143050666942E-016
-2.7125865266862269E-015	-1.9161909496628406E-014	-9.0497633840026565E-015	2.9172982750136132E-015	2.8053688746418353E-015
7.4300470103035744E-016	-7.4198630210625444E-015	-1.0588007397354777E-014	1.9593967230372657E-014	-1.8459936056455036E-014
1.3275865942935652E-014	6.6607973515146150E-014	5.8374478530375291E-014	-1.6938118646654461E-014	-4.0018821901761902E-014

Table 3: $\Delta u = u_{xx} + u_{yy}$ from equation c with nr = 6, ns = 4.

Conclusions

In this homework, I'm supposed to implement the error vs grid size convergence program. With the *differentiate.f90* file we are provided, we should use it to approximate the error resulting from different grid sizes we used. Here I tried three different mapping functions and three different equations (a, b, c) though the final results did not converge no matter I use error maximum or e_2. For the higher order implementation of differentiating, I can use the idea from $\Delta u = u_{xx} + u_{yy}$ and input the (n-1).th order to the original equation to find n.th order. Different mapping and equations have an effect of the final results, but the error plot should still be parallel with the corresponding h^2 blue line.