Homework 7: Ritz-Galerkin method for the one-dimensional Poisson-equation

Nelson Martin Sanchez Martinez

German Research School for Simulation Sciences GmbH
Numerical Methods for Partial Differential Equations
nelson.sanchez@rwth-aachen.de

Abstract: In this paper is discussed an implementation to solve a 1 dimensional Poisson equation problem with homogeneous boundary conditions using the Ritz-Galerkin method with a varying uni-dimensional grid. The solution is evaluated for 10, 50 and 100 grid points and the error in the maximum norm is evaluated to: 0.029635, 0.001183 and 0.000296 respectively. Then the solution is compared with the same problem solved with a constant spacing grid to conclude that with the graded mesh one order more of convergence is attained. Thus, the choice of grid makes the Ritz-Galerkin second order convergent

1 The implementation

1.1 The Ritz-Galerkin method

The problem for which this implementation was created has the following definition:

$$\begin{aligned} -u'' &= f, \qquad x \in (0,1) \\ u(0) = u(1) &= 0, \end{aligned}$$

Determine the right-hand side, such that the solution to this problem is given by $u(x) = 10(x^\alpha - x)$. Convince yourself that $u \in H^1(0,1)$ for $\alpha > 0.5$.

Construct a grid $0 = x_0 < x_1 < \cdots < x_N = 1$, where

$$x_j = \left(\frac{j}{N}\right)^{\frac{3}{2(\alpha-0.5)}} \tag{1}$$

Thus $[0,1] = \bigcup_{i=1}^{N} I_i$ where $I_i = [x_i, x_{i-1}]$. Solve the problem using the Ritz-Galerkin method and the finite dimensional space

$$S_h = \{v \in C([0,1]) : \ v|_{I_i} \in \mathcal{P}^1(I_i), i = 1, \ldots, N \,;\ v(0) = v(1) = 0\}. \tag{2}$$

Use a nodal basis $\{\phi_k\}_{k=1,\ldots,N-1}$ (i.e. basis functions $\phi_k \in S_h$, satisfying $\phi_k(x_i) = \delta_{ik}$ for $k = 1, \ldots, N-1$ and $i = 0, \ldots, N$).

Image 1: Set of conditions for this particular problem, it is worth to mention that f shows some issues with its second derivative, as is not defined for $x = 0$.

As usual we define the functions that are needed for our solution to work. First our spacing generating function, then the f declared in the problem definition. As we know that we'd have to solve a tri-diagonal system of equations of the shape $Ax = F$ at the end of the day, we define the Thomas algorithm as our weapon of choice. Later the A matrix of the system will be defined and last but not least, the results vector assembled (our F is called RHS).

Listing 1: Pyhton, Function definition

```
def spacing(j,N,alpha):
    if flag == 1:
        return  float(((j/N)**(3/(2*(alpha - 0.5)))))
    else:
        return x[j]

def g(x): ##Let's remember that f is the second spatial derivative of u(x
   )
    return float(10*(alpha -1)*alpha*(x**(alpha-2)))

def f(x): ## The solution function 10(x^alpha-x)
   return 10*(x**alpha - x)

def thomas(a,b,c,r):
##Thomas takes vectors and returns a vector of solutions
#Initializate the holder vectors
    c_star = np.linspace(0,0,N)
    d_star = np.linspace(0,0,N)
    u = np.linspace(0,0,N+1)

    c_star[0] = c[0] / b[0]
    d_star[0] = rhs[0]  / b[0]

    for j in range(1,N-2):
        c_star[j] = c[j]  / (b[j]-(a[j]*c_star[j-1]))
    for j in range(1,N-1):
        d_star[j] = (rhs[j] - a[j]*d_star[j-1]) / (b[j] - (a[j]*c_star[j
            -1]))
             #Backsubstitution
    u[N] = d_star[N-1]
    for j in range(N-1,-1,-1):
        u[j] = (d_star[j] - c_star[j]*(u[j+1]))
    return u
```

Now to build the "stiffness matrix A" the shape functions where operated to create the matrix $A_{i,j}$. Since we know this kind of method gives rise to a tri-diagonal matrix, then only these diagonals were investigated (the positions that are left are zero's everywhere). For the main diagonal $A_{i,i}$ we obtain:

$$A_{i,i} = \frac{1}{(x_{i+1}-x_i)} + \frac{1}{(x_i-x_{i-1})} \qquad (3)$$

Eq. 3 represents the interaction of both shape functions and their contributions to the system.

$$A_{i,i+1} = \frac{-1}{(x_{i+1}-x_i)} \qquad (4)$$

Eq. 4 represents the left contribution of the shape functions to the system.

$$A_{i,i-1} = \frac{-1}{(x_i-x_{i-1})}) \qquad (5)$$

Finally, Eq. 5 this represents the left contribution of the shape functions to the system.

Listing 2: Pyhton, Stiffness matrix assembly

```
## Create the grid first, it would be helpful for the graphs
    if flag ==1:
        for j in range(0,N+1):
            x[j] =  spacing(j,N,alpha)
    ##Initializate A
    A = np.zeros((N-1,N-1))
    ## Create the A matrix by feeding it grid points
    for j in range(0,N-1):
        fidx = j +1
        ##Not necessary to define this mesh, but it's more illustrative
           to have it here. However if memory is an issue, the matrix can
            be avoided and
        ## vectors a, b, c given directly to the Thomas function.
        A[j][j] = ((1 / ((spacing(fidx,N,alpha)) - (spacing(fidx-1,N,
           alpha)))) + (1 / ((spacing(fidx+1,N,alpha)) - (spacing(fidx,N,
           alpha)))))
        b[j] = A[j][j]

    for j in range(0,N-2):
        fidx = j +1
        A[j][j+1] = - (1 / ((spacing(fidx+1,N,alpha)) - (spacing(fidx,N,
           alpha))))
        c[j] = A[j][j+1]

    for j in range(1,N-1):
        fidx = j +1
        A[j][j-1] = -(1 / ((spacing(fidx,N,alpha)) - (spacing(fidx-1,N,
           alpha))))
        a[j] = A[j][j-1]
```

The right hand side of the equation is given by the trapezoidal integration rule used over the product of the f function and the shape functions. The following expression is then obtained:

$$\text{rhs}_j = -(x_j - x_{j-1})f(x_j) - (x_{j+1} - x_j)f(x_j) \qquad (6)$$

$$\text{where } f = 10(\alpha - 1)\alpha(x^{\alpha-2}) \qquad (7)$$

Listing 3: Pyhton, Right Hand side assembly

```
    for j in range(0,N-1):
        fidx = j+1
        right = -(spacing(fidx+1,N,alpha) - spacing(fidx,N,alpha))*0.5*g(
           spacing(fidx,N,alpha))
        left = -(spacing(fidx,N,alpha) - spacing(fidx-1,N,alpha))*0.5*g(
           spacing(fidx,N,alpha))
        rhs[j] = right +  left
```

Running each test for 10 ,50 and 100 grid points we obtain the following results:

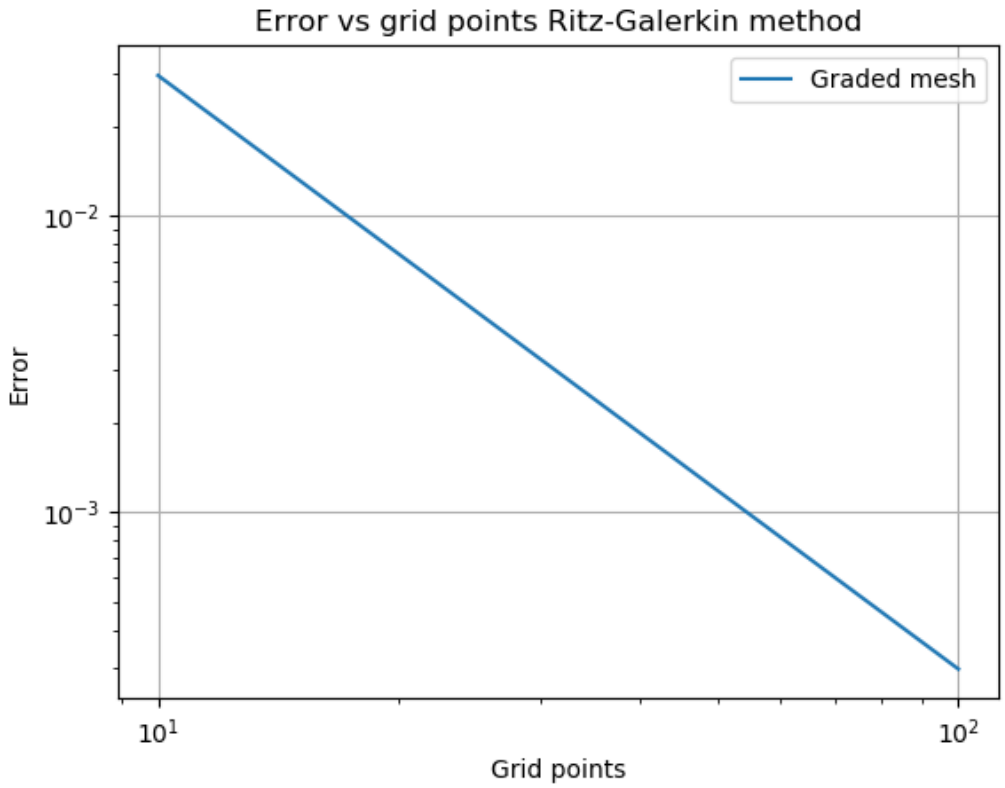

Image 2: Convergence study for the graded mesh. We can see that, contrary to what's expected, it converges faster than first order.

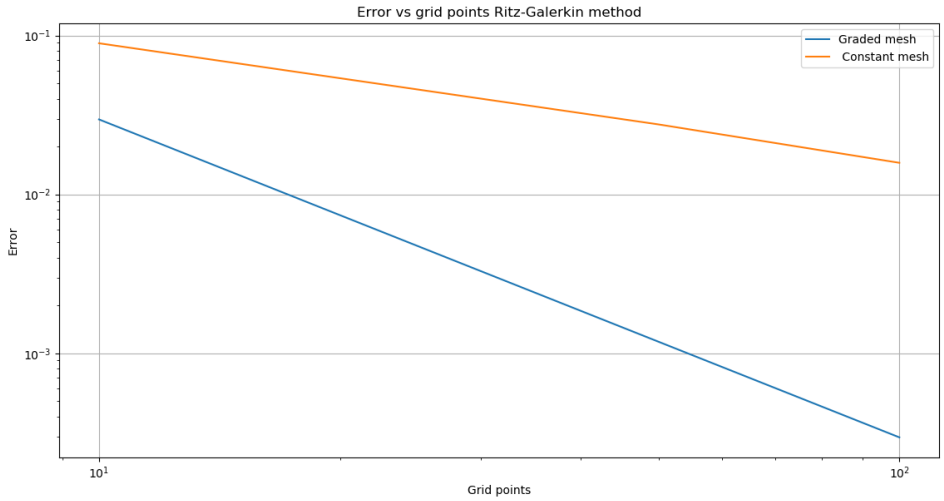

Image 3: Comparison of the convergence of the constant and graded mesh, here's clear how beneficial is to spend some effort in the meshing territory.

2 Conclusion

The results obtained after the convergence study are as follow:

Grid points	Varying mesh	Constant mesh
10	0.029635	0.094190
50	0.001183	0.027574
100	0.000296	0.015803

In particular, for the constant mesh to look like the varying grading mesh at 50 grid points (meaning they've the same order of magnitude for the norm) you'll need at least 500 points. With 500 points we obtain an error norm of : 0.004186. To get the 100 precision of the graded mesh with the constant you'll need some amount of additional grid points in the order of 10^1 . Which happens to be true, the run of the constant mesh with 2750 points gives a maximum norm of 0.0001 which is in the order of 0.000296 this is the norm of 100 points of the graded mesh. It can be seen then by using the graded mesh an order of magnitude extra in convergence could be obtained. Considering this evidence it's concluded that additional precision can be gained by adapting the mesh to the problem and that with this choice of mesh a second order convergence is attained.

References

[1] Georg, May. Homework 7. ,pages 1, Aachen, Nordrhein-Westfalen, Germany, 2018. CATS