IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-5728, p-ISSN: 2319-765X. Volume 14, Issue 2 Ver. II (Mar. - Apr. 2018), PP 55-60
www.iosrjournals.org

A Generalization on the Solution of the Second Order Ordinary Linear Boundary Value Problem by the Finite Element Method

M. R. Hossain[1,*], Touhid Hossain[2] , Md. Mamun-Ur-Rashid Khan[3]
[1]Department of Applied Mathematics, [2,3]Department of Mathematics
University of Dhaka,Dhaka-1000, Bangladesh
** Corresponding Author: M. R. Hossain*

Abstract: *In this paper we introduce a generalized MATHEMATICA code for solving one dimensional second order ordinary boundary value problems by Modified Galerkin's technique of finite element methodusing the standard basis functions of different degrees and any number of elements.*

Keywords:*Finite element method,Modified Galerkin's method, Standard shape function,MATHEMATICA*

Date of Submission: 09-04-2018 Date of acceptance: 23-04-2018

I. Introduction

The second order ordinary linear differential equations (ODE) arise in many applications including the spring-mass system [1], the string theory [2] and the general wave mechanics [3].To solve such ODE's, a number of different mathematical techniques, such as, the finite difference method (FDM), finite element methods (FEM) and the control volume method (CVM), have been used [4-7]. Depending on the physical problems, these equations, may be appeared in the general formor in the self-adjoint form [1], may be required different types of spatial grid schemes, or may be accompanied by different types of boundary conditions. For example, in the FEM, the choices of the types of the number of the elements or degree of the elements are very crucial. If the grid schemes or the boundary conditions are required to modify within a simulation, one has to reformulate the whole solution procedure accordingly, which is laborious and subject to the extra computational cost. Also to get the better accuracy by increasing the number or order of the elements, the computational work becomes lengthy and time consuming. To overcome these difficulties, this paper aims to generalize the FEM for 1D boundary value problems. A set of MATHEMATICA modules have also been presented for further reference.

II. Generalization

Consider the general 2nd order one dimensional ordinary BVP

$$a_1(x)\frac{d^2u}{dx^2}+a_2(x)\frac{du}{dx}+a_3(x)u=a_4(x),\ \ a\le x\le b \tag{1}$$

By converting which can also be written as the self-adjoint form as

$$-\frac{d}{dx}\left[p(x)\frac{du}{dx}\right]+\ q(x)u(x)=r(x),\ \ a\le x\le b. \tag{2}$$

The boundary conditions are

$$u(a)=\gamma_1\ ,\ \ u(b)=\gamma_2 \tag{1a}$$

$$\text{or, } u(a)=\gamma_3\ ,\ \ u'(b)=\gamma_4 \tag{1b}$$

$$\text{or, } u'(a)=\gamma_5\ ,\ \ u(b)=\gamma_6 \tag{1c}$$

$$\text{or, } u'(a)=\gamma_7\ ,\ \ u'(b)=\gamma_8 \tag{1d}$$

By taking the trial solution of the form

$$\tilde{u}(x)=\sum_{j=1}^{n}a_jN_j(x)$$

And using Modified Galerkin's [4-6] formulation we found the general system of equations

$$\begin{bmatrix} K_{11} & \dots\dots\dots\dots & K_{1n} \\ \dots\dots & \dots\dots\dots\dots & \dots\dots \\ K_{n1} & \dots\dots\dots\dots & K_{nn} \end{bmatrix} \begin{bmatrix} a_1 \\ \dots. \\ a_n \end{bmatrix} = \begin{Bmatrix} F_1 \\ \dots \\ F_2 \end{Bmatrix} \tag{3}$$

where,

$$K_{i,j} = \int_{[e]} \frac{dN_i}{dx}\, p(x) \frac{dN_j}{dx}\, dx + \int_{[e]} N_i(x)\, q(x) N_j(x)\, dx \ , and$$

$$F_i = \int_{[e]} r(x) N_i(x) dx + \left[\left(p(x)\frac{d\tilde{u}}{dx}\right) N_i(x)\right]_{[e]}.$$

The matrices $K = [K_{i,j}]$ and $F = [F_i]$ are known as Stiffness matrix and local force term [1], respectively.
Here, $N_j(x), j = 1, 2, \dots\dots$ are the standard shape functions of different order. For example, the linear shape functions are $N_1(\xi) = \frac{1-\xi}{2}, N_2(\xi) = \frac{1+\xi}{2}$, the quadratic shape functions are $N_1(\xi) = \frac{\xi}{2}(\xi - 1)$, $N_2(\xi) = 1 - \xi^2$, $N_3(\xi) = \frac{\xi}{2}(\xi + 1)$ and so on.
If we use the linear shape functions and 4 elements to solve a problem of the form (2) with any boundary condition the stiffness matrix becomes a 5×5 matrix and the local force term consists of 5 elements, similarly if we use linear shape functions and 8 elements then the stiffness matrix becomes a 9×9 matrix. Also if we use quadratic shape functions and 3 quadratic elements the stiffness matrix becomes a 7×7 matrix.From the above observations we can conclude the following lemma as:
Lemma: If we consider$m$ elements and $(n-1)^{th}$degree shape functions, then we have to calculate $(m \times n - m+1 \times (m \times n - m + 2)$ integral values to obtain matrices K and F.

III. Program Modules

The MATHEMATICA [8-10] code that we develop in this paper initially takes the values of the boundaries, degree of the shape functions and the number of elements as inputs. If the given differential equation is in the general form (1), then the code will convert it to self –adjointform (2). Nextthe code will generate the standard shape functions, as require, and compute the entries of the matrices K and F by taking the boundary conditions (essential or suppressible) given. Finally, it shows the calculated numerical results with absolute error (if the exact solution is available) in the tabular form and represents the results graphically.

IV. Numerical Examples

Example 4.1:
Consider the following BVP [1]

$$\frac{d}{dx}\left(x\frac{du}{dx}\right) = \frac{2}{x^2}, \quad 1 < x < 2, \quad u(1) = 2, \ u'(2) = -\frac{1}{4}$$

Input:

$$\boldsymbol{a = 1}, \quad \boldsymbol{b = 1}, \quad \boldsymbol{p(t) = -t}, \quad \boldsymbol{q(t) = 0}, \quad \boldsymbol{r(t) = \frac{2}{t^2}}, \quad \boldsymbol{u(1) = 2}, \quad \boldsymbol{u'(2) = -\frac{1}{4}}$$

$$\boldsymbol{u_{exact} = \frac{2}{t} + \frac{1}{2}\ln t}$$

Output:
The matrices K and $F$are:

$$K = \begin{bmatrix} -9/2 & 9/2 & 0 & 0 & 0 \\ 9/2 & -10 & 11/2 & 0 & 0 \\ 0 & 11/2 & -12 & 13/2 & 0 \\ 0 & 0 & 13/2 & -14 & 15/2 \\ 0 & 0 & 0 & 15/2 & -15/2 \end{bmatrix}; \quad F = \begin{bmatrix} 2 + du[1] - 8\,Log[\frac{5}{4}] \\ -8\,Log\left[\frac{6}{5}\right] + 8\,Log[\frac{5}{4}] \\ -8\,Log\left[\frac{7}{6}\right] + 8\,Log[\frac{6}{5}] \\ -8\,Log\left[\frac{8}{7}\right] + 8\,Log[\frac{7}{6}] \\ -1 - 2\,du[2] + 8\,Log[\frac{8}{7}] \end{bmatrix}$$

Table 1: Results using Linear Shape function and 4 elements:

Nodes	Approx. Solution	Exact Solution	Absolute Error
1.0	2	2	0
1.25	1.71441	1.71157	0.00283969
1.5	1.54013	1.53607	0.00405968
1.75	1.42732	1.42267	0.00465969
2.0	1.35156	1.34657	0.00498432

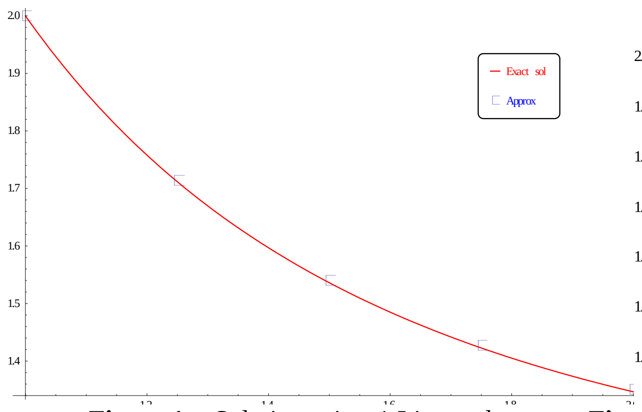
Table 2: Results using Quadratic Shape function and 4 elements:

Nodes	Approx. Solution	Exact Solution	Absolute Error
1.0	2	2	0
1.125	1.83673	1.83667	0.0000566035
1.25	1.71159	1.71157	0.0000148841
1.375	1.61381	1.61377	0.0000343892
1.5	1.53609	1.53607	0.0000192144
1.625	1.47355	1.47352	0.0000272429
1.75	1.42269	1.42267	0.0000207598
1.875	1.381	1.38097	0.000024504
2.0	1.34659	1.34657	0.0000213972

Table 3: Results using Cubic Shape function and 4 elements:

Nodes	Approx. Solution	Exact Solution	Absolute Error
1.0	2	2	0
1.08333	1.88616	1.88618	0.0000197765
1.16667	1.79134	1.79136	0.0000201209
1.25	1.71157	1.71157	6.50232×10^{-8}
1.33333	1.64383	1.64384	7.07551×10^{-6}
1.41667	1.58591	1.58592	7.17107×10^{-6}
1.5	1.53607	1.53607	7.77409×10^{-8}
1.58333	1.49292	1.49292	2.96977×10^{-6}
1.66667	1.45541	1.45541	3.00205×10^{-6}
1.75	1.42267	1.42267	8.10082×10^{-8}
1.83333	1.39398	1.39398	1.38426×10^{-6}
1.91667	1.36877	1.36877	$1.39677\times10-^{6}$
2.0	1.34657	1.34657	8.20266×10^{-8}

Graphical Results:



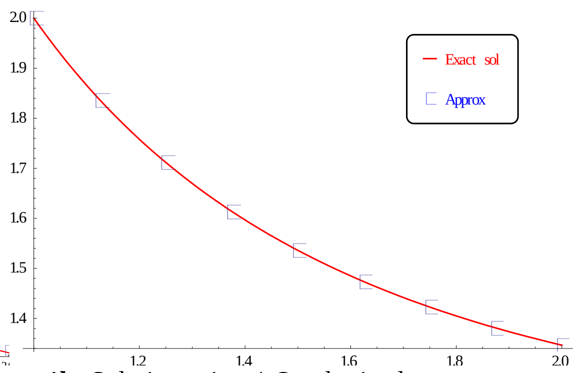


Figure 1a: Solution using 4-Linear elements **Figure 1b:** Solution using 4-Quadratic elements

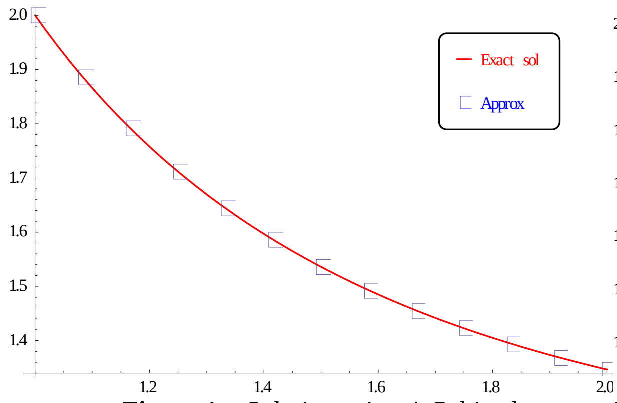


Figure 1c: Solution using 4-Cubic elements

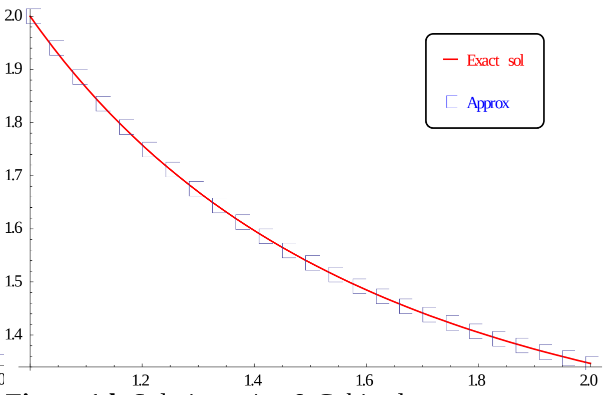


Figure 1d: Solution using 8-Cubic elements

Example 4.2:
Consider the BVP [11]

$$\frac{d^2u}{dx^2}+\frac{4}{x}\frac{du}{dx}+\frac{2}{x^2}u=\frac{2}{x^2}\ln x,\quad 1<x<2,\qquad u(1)=\frac{1}{2},\qquad u(2)=\ln 2$$

Input:

$$\boldsymbol{a=1, b=2,\ a_2(t)=1,\ a_1(t)=\frac{4}{t},\ a_0(t)=\frac{2}{t^2},\qquad cc(t)=\frac{2}{t^2}\ln t, u(1)=\frac{1}{2},\ \ u(2)=ln\,2}$$

$$\boldsymbol{u_{exact}=\frac{4}{t}-\frac{2}{t^2}+\ln t-\frac{3}{2}}$$

Graphical Results:

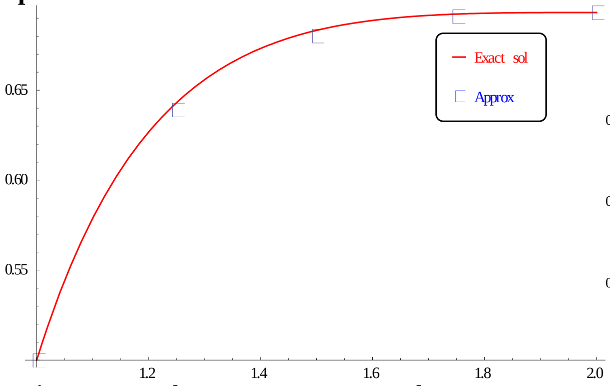


Figure 2a: Solution using 4-Linear elements

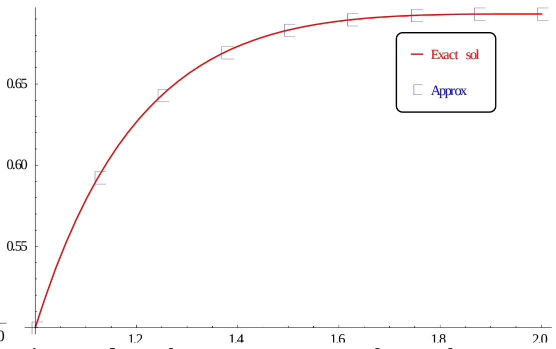


Figure 2b: Solution using 4-Quadratic elements

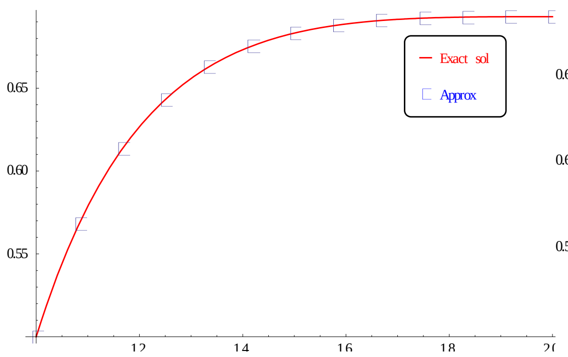


Figure 2c: Solution using 4-Cubic elements

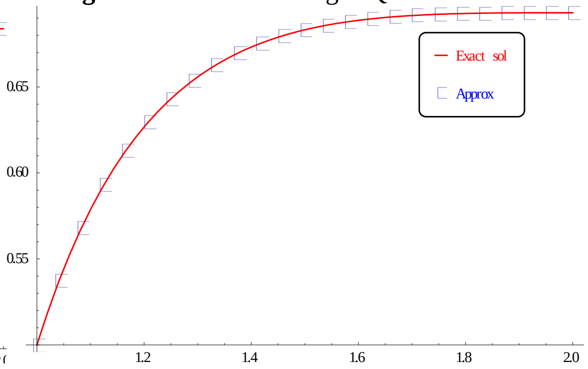


Figure 2d: Solution using 8-Cubic elements

V. Discussion

In example 4.1, we see that if we take 4 elements and use linear shape functions we get the approximate solutions (node wise) with percentage errors within $\pm 0.1\%$ to $\pm 0.3\%$ (Table 1). Whereas, if we consider 4 elements with quadratic shape functions (although it requires tiresome hand calculations) we get the approximate solutions with percentage errors within $\pm 0.001\%$ to $\pm 0.003\%$ (Table 2). With the same number of elements and cubic shape functions we get more accurate results (Table 3).

VI. Conclusion

To get better approximate solutions (with graphical representation and error analysis, if exact solution is available) of any form of second order BVP's by using Modified Galerkin's technique of FEM with any number of elements and standard shape functions of any degrees by avoiding time consuming hand calculations (impossible in some cases), we can use the MATHEMATICA code that we developed in this paper.

References

[1]. Lewis, P. E., Ward J. P., 1991, *The Finite Element Method*, Addison-Wesley Publishers Ltd., Great Britain.

[2]. Fish J., Belytschko T., 2007, *A First Course in Finite Elements*, John Wiley & Sons Ltd, England.

[3]. Reddy J. N., 1993, *An Introduction to The Finite Element Method*, MacGraw-Hill Inc.

[4]. Smith I. M., Griffiths D. V., 1982, *Programming The Finite Element Method*, John Wiley & Sons Ltd, England.

[5]. Zienkiewicz O.C., Taylor R.L., 2000, *The Finite Element Method*, Butterworth-Heinemann, England.

[6]. Gentian Zavalani,*AGalerkin Finite Element Method for Two-Point Boundary Value Problems of Ordinary Differential Equations*. Applied and Computational Mathematics. Vol. 4, No. 2, 2015, pp. 64-68. doi: 10.11648/j.acm.20150402.15.

[7]. Petrolito J., *Approximate solutions of differential equations using Galerkin's method and weighted residuals*, International Journal of Mechanical Engineering Education Vol 28 No 1, pp. 14-26.

[8]. Don E., 2001, *Theory and Problems of Mathematica*, MacGraw-HillInc. USA.

[9]. Hartmut F. W. Höft, Margret Höft, 2003,*Computing with Mathematica*, Elsevier Science (USA).

[10]. Paul R. Wellin, Richard J. Gaylord, Samuel N. Kamin, 2005, *An Introduction to Programming with Mathematica*, Cambridge University Press, New York, USA.

[11]. Burden Richard L. &Douglas Fairs J., 2001, *Numerical Analysis*, Thomson Learning.

Appendix

Mathematica Code:

```
FEM:=Module[{
a=Input["Enter the value of a"],
b=Input["Enter the value of b"],
deg=Input["Enter the value of the degree of the standard shape function"],
m=Input["Enter the number ofelements"],
equform=Input["Enter \n 1 if the given differential equation is in general
form \n 2 if the given differential equation is in self-adjoint form"]
},
n=deg+1;h=2/deg;
If[equform□1,
a2[t_]=Input["Func. a2(t) of a2(t)u''+a1(t)u'+a0(t)u=cc(t)"];
a1[t_]=Input["Func. a1(t) of a2(t)u''+a1(t)u'+a0(t)u=cc(t)"];
a0[t_]=Input["Func. a0(t) of a2(t)u''+a1(t)u'+a0(t)u=cc(t)"];
cc[t_]=Input["Func. cc(t) of a2(t)u''+a1(t)u'+a0(t)u=cc(t)"];
p[t_]=Exp[Integrate[a1[t]/a2[t],t]];
auxfun[t_]=-a2[t]/p[t];q[t_]=a0[t]/auxfun[t];r[t_]=cc[t]/auxfun[t],
If[equform□2,
p[t_]=Input["Func. p[t] of -(p(t)u')'+q(t)u=r(t)"];
q[t_]=Input["Func. q[t] of -(p(t)u')'+q(t)u=r(t)"];
r[t_]=Input["Func. r[t] of -(p(t)u')'+q(t)u=r(t)"]]];
For[i=1,i≤m*n-(m-1),i++,z[i_]:= a+(i-1)*(b-a)/(m*n-m)];
For[i=1,i≤n,i++,x[i_]:= -1+(i-1)*h];
For[ j=1,j≤n,j++,
For[ i=1,i≤n,i++,lhs[i]:=Sum[α[c]*Product[x[i],{l,1,c}] ,{c,0,n-1}];
If[i□j,rhs[i]:=1 ,rhs[i]:=0 ]];
s1=Table[lhs[i]==rhs[i],{i,1,n}];
s2=Table[α[i],{i,0,n-1}];
s3=Solve[s1,s2];
sfun[j][ξ]=Sum[s3[[1,i+1,2]]*ξ^i,{i,0,n-1}];];
j=0;
For[l=1,l≤m*n-m+1,l++,F[l]=0;For[c=1,c≤m*n-m+1,c++,K[l,c]=0]];
For[i=1,i≤m,i++,y[ξ]=Sum[z[l+j]*sfun[l][ξ],{l,1,n}];
dxξ[ξ]=D[y[ξ],ξ];dξx[ξ]=1/dxξ[ξ];
For[l=1,l≤n,l++,
If[l□1,F[l+j]=F[l+j]+Integrate[r[y[ξ]]*sfun[l][ξ]*dxξ[ξ],{ξ,-1,1}]-
p[z[1+j]]*du[z[1+j]],
If[l□n,F[l+j]=F[l+j]+Integrate[r[y[ξ]]*sfun[l][ξ]*dxξ[ξ],
{ξ,-
1,1}]+p[z[n+j]]*du[z[n+j]],F[l+j]=F[l+j]+Integrate[r[y[ξ]]*sfun[l][ξ]*dxξ[ξ
],{ξ,-1,1}]]];
For[c=1,c≤n,c++,
K[l+j,c+j]=K[l+j,c+j]+Integrate[p[y[ξ]]*D[sfun[l][ξ],ξ]*D[sfun[c][ξ],ξ]*dξx[
ξ],{ξ,-1,1}] + Integrate[q[y[ξ]]*sfun[l][ξ]*sfun[c][ξ]*dxξ[ξ],{ξ,-1,1}]]];
j=j+n-1;];

g=Input["input
\n 1 if both boundary conditions are essential
\n 2 if last condition is  suppresible
\n 3 if first condition is  suppresible
\n 4 if both boundary conditions are suppresible"];
anasol=Input["Enter \n 1 if analytical solution is known \n 0 if analytical
solution is unknown"];
If[g□1,u[a]=Input["u[a]"];u[b]=Input["u[b]"];
s4=LinearSolve[Table[K[i,j]//N,{i,2,m*n-m},{j,2,m*n-m}],
Table[(F[i]-u[a]*K[i,1]-u[b]*K[i,m*n-m+1])//N,{i,2,m*n-m}]];
asol[1]=u[a];asol[m*n-m+1]=u[b];
For[i=2,i≤ m*n-m,i++,asol[i]=s4[[i-1]]],
If[g□2,u[a]=Input["u[a]"];β=Input["u'[b]"];
s4=LinearSolve[Table[K[i,j]//N,{i,2,m*n-m+1},{j,2,m*n-m+1}],
Table[((F[i]/.du[b]→β)-u[a]*K[i,1])//N,{i,2,m*n-m+1}]];
asol[1]=u[a];For[i=2,i≤ m*n-m+1,i++,asol[i]=s4[[i-1]]],
If[g□3,u[b]=Input["u[b]"];β=Input["u'[a]"];
s4=LinearSolve[Table[K[i,j]//N,{i,1,m*n-m},{j,1,m*n-m}],
Table[((F[i]/.du[a]→β)-u[b]*K[i,m*n-m+1])//N,{i,1,m*n-m}]];
asol[m*n-m+1]=u[b];For[i=1,i≤ m*n-m,i++,asol[i]=s4[[i]]],
If[g□4,β=Input["u'[a]"];γ=Input["u'[b]"];
s4=LinearSolve[Table[K[i,j]//N,{i,1,m*n-m+1},{j,1,m*n-m+1}],
Table[F[i]/.{du[a]→β,du[b]→γ}//N,{i,1,m*n-m+1}]];
For[i=1,i≤ m*n-m+1,i++,asol[i]=s4[[i]]]]]]];
If[anasol□1,f[t_]=Input["Analytical solution u(t) in variable t"]];
Print["The matrices K and F are: \n"];
Print["K=",MatrixForm[Table[N[K[i,j]],{i,1,m*n-m+1},{j,1,m*n-m+1}]]];
Print["F=",MatrixForm[Table[N[F[i]],{i,1,m*n-m+1}]]];
If[anasol□1,Print[TableForm[Table[{N[z[i]],N[asol[i]],f[N[z[i]]],
error[i]=Abs[f[N[z[i]]]-asol[i]]}, {i,1,m*n-m+1}],
TableHeadings→{None,{"Nodes","Approx.
Solutions","Exactsolutions","Error"}},TableAlignments→Left]];

Show[{Plot[f[t],{t,a,b},
PlotStyle→Red],ListPlot[Table[{N[z[i]],asol[i]},{i,1,m*n-m+1}],
PlotMarkers→{"ノ",15}]},Epilog→Inset[Framed[Column[{LineLegend[
{Red},{"Exact soln"},LegendMarkerSize→10,LabelStyle→Red],
PointLegend[{Blue},{"Approx. Soln"},LegendMarkers→{"ノ"},
LegendMarkerSize→10,LabelStyle→Blue]}],RoundingRadius→9],
Scaled[{.8,.7}]]],

TableForm[Table[{N[z[i]],asol[i]},{i,1,m*n-m+1}],
TableHeadings→{None,{"Nodes","Approx. Solutions"}},
TableAlignments→Left]]
]
FEM
```