A computer program for approximating a linear operator equation using a generalized Fourier series

M. G. SEIBEL, A. F. LEAL, M. R. BARTON

Department of Mathematics, California State University, Fullerton, CA 92634, USA

THEODORE V. HROMADKA II

Williamson and Schmid, Irvine, California 92714, USA

Many important engineering problems fall into the category of being linear operators, with supporting conditions. In this paper, an inner-product and norm is used which enables the numerical modeler to approximate such by developing a generalized Fourier series. The resulting approximation is the "best" approximation in that a least-squares (L^2) error is minimized simultaneously for fitting both the problem's boundary conditions and satisfying the linear operator relationship (the governing equations) over the problem's domain (both space and time). Because the numerical technique involves a well-defined inner-product, error evaluation is readily available using Bessel's inequality. Minimization of the approximation error is subsequently achieved with respect to a weighting of the inner product components, and the addition of basis functions used in the approximation. A computer program source code is provided (see Appendix A) to implement the procedures.

AN INNER PRODUCT FOR THE SOLUTION OF LINEAR OPERATOR EQUATIONS

The general setting for solving a linear operator equation with boundary values by means of an inner product is as follows: let Ω be a region in R^m with boundary Γ and denote the closure of Ω by $cl(\Omega)$. Consider the Hilbert space $L^2(cl(\Omega), d\mu)$. which has inner products $(f, g) = \int fg\, d\mu$. One way to define the necessary inner product for the development of a generalized Fourier series is to let μ be one measure μ_1 on Ω and another measure μ_2 on Γ. Once choice for a plane region would be for μ_1 to be the usual two-dimensional Lebesgue measure $d\Omega$ on Ω and for μ_2 to be the usual arc length measure $d\Gamma$ on Γ. Then, an inner product is given by[1,2]:

$$(f, g) = \int_{\Omega} fg\, d\Omega + \int_{\Gamma} fg\, d\Gamma \tag{1}$$

Paper accepted February 1990. Discussion closes January 1992.

Consider a boundary value problem consisting of an operator L defined on domain $D(L)$ contained in $L^2(\Omega)$ and mapping into $L^2(\Omega)$, and a boundary condition operator B defined on a domain $D(B)$ in $L^2(\Omega)$ and mapping it into $L^2(\Gamma)$. The domains of L and B have to be chosen so at least for f in $D(L)$, Lf is in $L^2(\Omega)$, and for f in $D(B)$, Bf is in $L^2(\Gamma)$. For example we could have $Lf = \nabla^2 f$, and $Bf(s)$ equal the almost everywhere (a.e.) radial limit of f at the point s on Γ, with appropriate domains.

The next step is to construct an operator T mapping its domain $D(T) = D(L) \cap D(B)$ into $L^2(cl(\Omega))$ by[3]:

$$\begin{aligned} Tf(x) &= Lf(x) \text{ for } x \text{ in } \Omega \\ Tf(s) &= Bf(s) \text{ for } s \text{ on } \Gamma \end{aligned} \tag{2}$$

From equation (2) there exists a single operator T on the Hilbert space $L^2(cl(\Omega))$ which incorporates both the operator L and the boundary conditions B, and which is linear if both L and B are linear.

Consider the inhomogeneous equation $LF = g_1$ with the inhomogeneous boundary conditions $Bf = g_2$. Then define a function g on $cl(\Omega)$ by:

$$g = g_1 \text{ on } \Omega$$

$$g = g_2 \text{ on } \Omega$$

Then if the solution exists for the operator equation:

$$Tf = g$$

the solution f satisfies $\nabla^2 f = g_1$ on Ω, and $f = g_2$ on Γ in the usual sense of meaning that the radial limit of f is g_2 on Γ. One way to attempt to solve the equation $Tf = g$ is to look at a subspace D_n of dimension n, which is contained in $D(T)$, and to try to minimize $\| Th - g \|$ over all the h in D_n such as developed in detail in Hromadka *et al.*[2,4]

DEFINITION OF INNER-PRODUCT AND NORM

Given a linear operator relationship:

$$L(\phi) = h \text{ on } \Omega,\ \phi = \phi_b \text{ on } \Gamma \tag{3}$$

defined on the problem domain Ω with auxiliary conditions of $\phi = \phi_b$ on the boundary Γ. Here Ω may represent both time and space, and ϕ_b may be both initial and boundary conditions.

Choose a set of m linearly independent functions $\langle f_j \rangle^m$ and let S^m be the m-dimensional space spanned by the elements of $\langle f_j \rangle^m$. Here, the elements of $\langle f_j \rangle^m$ will be assumed to be functions of the dependent variables appearing in equation (3).

An inner-product is defined for elements of S^m by (u, v) where for $u, v \in S^m$

$$(u, v) = \int uv \, d\Gamma + \int LuLv \, d\Omega \tag{4}$$

It is seen that (u, v) is indeed an inner-product[2]. A normal '$\| \; \|$' immediately follows from (4) by

$$\| u \| \equiv (u, v)^{1/2} \tag{5}$$

The generalized Fourier series approach can now be used to obtain the 'best' approximation $\phi_m \in S^m$ of the function ϕ using the newly defined inner-product and corresponding norm presented in equations (4) and (5).

The next step in developing the generalized Fourier series is to construct a new set of functions $\langle g_j \rangle^m$ which are the orthonormal representation of the $\langle f_j \rangle^m$.

ORTHONORMALIZATION PROCESS

The functions $\langle g_j \rangle^m$ can be obtained by the well-known Gramm-Schmidt procedure[4] using the newly defined norm of equation (4). That is:

$$g_1 = f_1 / \| f_1 \|$$
$$\vdots$$
$$g_m = [f_m - (f_m, g_1)g_1 - \ldots - (f_m, g_{m-1})g_{m-1}] / \| f_m - (f_m, g_1)g_1 - \ldots - (f_m, g_{m-1})g_{m-1} \|$$

Hence, the elements of $\langle g_j \rangle^m$ satisfy the convenient properties that

$$(g_j, g_k) = \begin{cases} 0, \text{ if } j \neq k \\ 1, \text{ if } j = k \end{cases} \tag{7}$$

The elements $\langle g_j \rangle^m$ also form a basis for S^m but, because of equation (7), can be directly used in the development of a generalized Fourier series where the computed coefficients do not change as the dimension m of $\langle g_j \rangle^m$ increases. Each element $\phi_m \in S^m$ can now be written as

$$\phi_m = \sum_{j=1}^{m} \gamma_j g_j, \; \phi_m \in S^m \tag{8}$$

where γ_j are unique real constants.

GENERALIZED FOURIER SERIES

The ultimate objective is to find the element $\phi_m \in S^m$ such that $\| \phi_m - \phi \|$ is a minimum. That is, we want $\| \phi_m - \phi \|^2$ to be a minimum, where

$$\| \phi_m - \phi \|^2 = \int_\Gamma \left(\sum_{j=1}^{m} \gamma_j g_j - \phi_b \right)^2 d\Gamma + \int_\Omega \left(L \sum_{j=1}^{m} \gamma_j g_j - L \right)^2 d\Omega \tag{9}$$

Remembering that L is a linear operator, and $L\phi = f$ by the problem definition of equation (3),

$$\| \phi_m - \phi \|^2 = \int_\Gamma \left(\sum_{j=1}^{m} \gamma_j g_j - \phi_b \right)^2 d\Gamma + \int_\Omega \left(\sum_{j=1}^{m} \gamma_j Lg_j - f \right)^2 d\Omega \tag{10}$$

Thus, minimizing $\| \phi_m - \phi \|^2$ is equivalent to minimizing the error or approximating the boundary conditions and the error of approximating the governing operator relationship in a least-square (or L^2) sense. Because the $\langle g_j \rangle^m$ are orthonormalized and the inner-product $(,)$ is well-defined, the coefficients γ_j of equation (8) are immediately determined by the generalized Fourier constants, γ_j^*, where:

$$\gamma_j^* = (g_j, \phi), \; j = 1, 2, \ldots, m \tag{11}$$

Thus

$$\phi_m^* = \sum_{j=1}^{m} \gamma_j^* g_j = (g_j, \phi) g_j \tag{12}$$

is the 'best' approximation of ϕ, in the space S^m.

APPROXIMATE ERROR EVALUATION

Due to the generalized Fourier series approach and the definition of the inner-product. Bessel's inequality applies. That is, for any dimensions m:

$$(\phi, \phi) \geqslant \sum_{j=1}^{m} (g_j, \phi)^2 = \sum_{j=1}^{m} \gamma_j^{*2} \tag{13}$$

where

$$(\phi, \phi) = \int_\Gamma (\phi)^2 d\Gamma + \int_\Omega (L\phi)^2 \, d\Omega = \int_\Gamma \phi^2 \, d\Gamma + \int_\Omega f2 \, d\Omega \tag{14}$$

Equation (14) is readily evaluated and forms an upper bound to the sum of $(g_j, \phi)^2$ as the dimension m increases. Consequently, one may interact with the approximation effort by carefully adding functions to the $\langle g_j \rangle^m$ in order to best reduce the difference computed by Bessel's inequality.

THE WEIGHTED INNER PRODUCT

In the inner product of (4), equal weight is given to the various requirements imposed on the best approximation function ϕ_m from the space S^m spanned by the m linearly independent basis functions $\langle f_j \rangle^m$. Namely, the L^2 error in satisfying the linear operator relationship over Ω is considered by equal importance as the L^2 error in satisfying the problem's boundary (and initial) conditions.

Due to the limitations of computer power, only a finite number of basis functions can be used for approximation purposes, and so an argument is made to weight the terms which compose the inner product differently. For $0 < \varepsilon < 1$, one weighting of equation (4) is simply:

$$(u, v) = \varepsilon \int_\Gamma uv \, d\Gamma + (1 - \varepsilon) \int_\Omega LuLv \, d\Omega \tag{15}$$

In equation (15), an ε–value close to 1 would force the

approximation function ϕ_n of S^m to focus upon satisfying the problem's boundary conditions rather than satisfying the linear operator. Similarly, an ε-value close to 0 would focus the ϕ_m approximation towards satisfying the linear operator relationship.

It is noted that equation (15) is still an inner product for a given choice of ε, and will be used to develop the generalized Fourier series using the previous presented procedures. And as the dimension S^m increases, the Bessel's inequality still applies in that $\chi = \chi_\varepsilon$, and

$$\chi_\varepsilon = 0 \Rightarrow \| \phi_m - \phi \| = 0 \tag{16}$$

USER INSTRUCTIONS FOR THE GENERALIZED FOURIER SERIES ANALYSIS PROGRAM

In this PASCAL implementation of the program the source code requires alteration and re-compilation for each different problem, and for each trial solution to a problem. The program structure remains the same, but the number of trial functions and the equations for computing the functions will change from problem to problem. The following is the procedure for tailoring the program to solve a specific problem, using the problem

$$y'' + y = 0,\ y(0) = 0,\ y(\pi/2) = 1$$

as an example.

The problem

Generally, the problem statement will be given in the form of a linear operator which describes the points within a boundary, and in the form of point values on the boundary. From the given set of conditions a vector, θ, is constructed. The elements of this vector represent the given points which will be used in the approximation. The user will then construct a set of basis vectors which will be used in the least squares approximation of θ. Suppose, we wish to solve our example problem using 10 points on the interval $(0, \pi/2)$. Then

$$\theta = 0, 1, 0, 0, \ldots, 0$$

where the first two elements, 0 and 1, are the values at the end points, 0 and $\pi/2$, and the remaining zeros represent the operator $y'' + y$ applied to 10 points in the interior. Using the GFSA program, θ will be approximated in the least-squares sense by a set of basis vectors constructed from functions chosen by the user.

The solution

With the problem represented by the vector θ, the user decides on the following parameters for his trial solution:

1. The number of interior points.
2. A weighting factor for the boundary points.
3. A set of basis functions.

The interior points are chosen depending on where the most accuracy is desired. The more points which are chosen in a certain area, the more accurate will the approximation be in that area. The weighting factor for the boundary points is also determined by accuracy considerations. The weighting factor, ε, is a number between 0 and 1 which scales the boundary point values. The factor $1 - \varepsilon$ then scales the interior points. If the user wants the approximation to be more accurate at the boundary points then he sets ε to a number close to 1, and if the interior points are more critical then he chooses ε to be a number near 0.

The user will choose basis functions depending on his insight into the problem; functions will be chosen which are believed to have the best chance of contributing to an accurate solution. Functions can be added and deleted to test their effect on the solution.

For our example problem $y'' + y = 0$, suppose that the trial functions 1, x, $\sqrt{x}$, $\sin x$, $\cos x$, $x\sin x$, and $x\cos x$ have been chosen. Then the basis vectors are constructed as follows:

$$\begin{aligned}
1 &: [1, 1, 1, 1, \ldots, 1] \\
x &: [0, \pi/2, x, x, \ldots, x] \\
\sqrt{x} &: [0, \sqrt{\pi}/2, 2 + x, \ldots, 2 + \sqrt{x}] \\
\sin x &: [0, 1, 0, 0, \ldots, 0] \\
\cos x &: [1, 0, 0, 0, \ldots, 0] \\
x\sin x &: [0, \pi/2, 2\cos x, \ldots, 2\cos x] \\
x\cos x &: [0, 0, -2\sin x, \ldots, -2\sin x]
\end{aligned}$$

The first element in each vector is the function evaluated at the first boundary point ($x = 0$), the second element in each vector is the function evaluated at the second boundary point ($x = \pi/2$), and the remaining elements are the results of the operator $y'' + y$ applied to the functions.

Adapting the program

Refer to the listing of the PASCAL program which solves the example problem $y'' + y = 0$, $y(0) = 0$, $y(\pi/2) = 1$, as an illustration of the following steps.

1. Input of Data

There are two methods of inputing data: one may create a data file which the program calls, or one may hardcode the data into the main program. The latter is advantageous in the case where the interior points are to be equally spaced.

In the case of hardcoding, follow the procedure below:

A) In the main program, assign values to the variables for the number of boundary points (nbon), the number of interior points (nint), and the weighting factor (epslo). Example:

```
nbon  : = 2;
nint  : = 10;
epslo : = 0.5;
```

B) In the main program, define the data points. Example of equally spaced interior points:

```
x[1]     : = 0;
x[2]     : = π/2;
value[1] : = 0;
value[2] : = 1;
ntol     : = nbon + nint;
for i    : = (nbon + 1) to ntol do
    begin
    x[i]: = x[nbon]*(i-nbon)/(nin + 1);
    value[i]: = 0;
    y[i]: = 0;
    end;
```

Note that the array y[ ] must be set to zeros since this is a two-dimensional problem.

In the case of creating a data file, the data file should comprise of the following lines:

line #	line contents
1)	nbon nint epsio
2)	x[i] y[i] value[i]
.	
.	
nbon + 1)	x[nbon + 1], y[nbon + 1],value[nbon + 1]
nbon + 2)	x[nbon + 2],y[nbon + 2],value[nbon + 2]
.	
.	
nbon + ntol)	x[nbon + ntol], y[nbon + ntol], value[nbon + ntol]

2. In the procedure 'basis' declare the variable to be used for the basis functions and the number of functions:

```
var
   xx, x2 : real;
being
   x2 : = sqr(xxx);
   nb : = 7; (7 functions)
```

3. In the procedure 'basis' define the solution, v(xx), at each point xx as the linear sum of the basis functions with coefficients xk[nb, j]:

```
v : =xk[nb, 1]*1 + xk[nb, 2]*xx + xk[nb, 3]*x2
                + xk[nb, 4]*sin(xx)
                + xk[nb, 5]*cos(xx)
                + xk[nb, 6]*xx*sin(xx)
                + xk[nb, 7]*xx*cos(xx);
```

4. In the procedure 'basis', define f as the linear operator applied to v; for the example problem this is $v'' + v$:

```
f: = xk[nb, 1] + 2*xk[nb, 3] + xk[nb, 2]*xx
                + xk[nb, 3]*x2
                + (xk[nb, 6]*xx - 2*xk[nb, 5])*sin(xx)
                + (xk[nb, 7]*xx + 2*xk[nb, 4])*cos(xx);
```

5. Define the elements of the basis vectors in the procedure 'basis'. Example:

```
if i > >nnod then
begin
      g[1, i]: = 1;
      g[2, i]: = xx;
      g[3, i]: = 2 + x2;
      g[4, i]: = 0;
      g[5, i]: = 0;
      g[6, i]: = 2*cos(xx);
      g[7, i]: = -2*sin(xx);
end
else
begin
      g[i,i]: = 1;
      g[2, i]: = xx;
      g[3, i]: = x2;
      g[4, i]: = sin[xx);
      g[5, i]: = cos(xx);
      g[6, i]: = xx*sin(xx);
      g[7, i]: = xx*cos(xx);
end;
```

6. If the exact solution to the problem is known, then the equation for the exact solution is defined of the main program in order to compute the approximation errors. Example:

```
ex: = sin(x[i]);
```

If there is no known exact solution then the error computations at the end of the program are meaningless and they can be deleted.

7. Compile the modified program.

When the program is run the outputs go to the printer with device name 'LPT1'. The information printed consists of the data points, the results of the orthogonality test, the coefficients for the least squares solution, and the data point values computed from the approximation function.

SUMMARY OF RESULTS

Given the boundary problem:

$$y^{(4)} + 2y'' + y = 0,\ y(0) = 1,\ y'(0) = 0,$$

$$y\left(\frac{\pi}{2}\right) = 0,\ y'\left(\frac{\pi}{2}\right) = 1,$$

an analytic solution is easily derived:

$$y(x) = 2.14 \sin x + \cos x - 214\ x \cos x - 1.363\ x \sin x$$

With a complete basis, the computer program provides the exact solution. The following figures demonstrate the effect of an incomplete basis. In both cases, $x\sin x$ has been deleted from the set of basis functions. Figs 1 and 2 demonstrate the effect of a higher order monomials being added to the set of basis functions. With the addition of each higher order monomial, the approximation becomes more accurate. Figs 3 and 4 demonstrate the

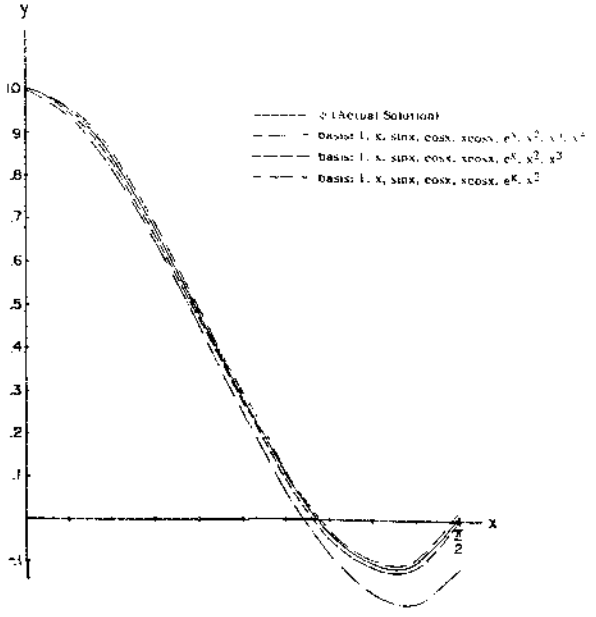

Fig. 1. $\tilde{\phi}$ for various basis

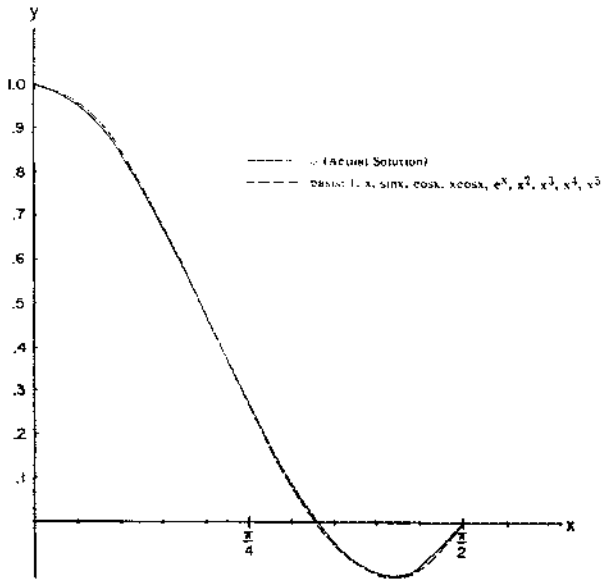

Fig. 2. $\tilde{\phi}$ for various basis

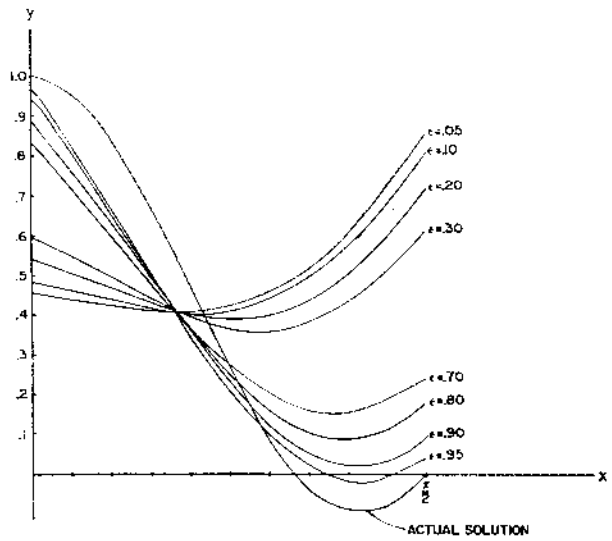

Fig. 3. Solution space for various ε (xsinx deleted from basis functions)

effect of a change in ε, i.e., focusing upon satisfying the problem's boundary conditions or upon satisfying the operator. As expected, a small ε satisfies the operator space well but gives a poor approximate solution, while

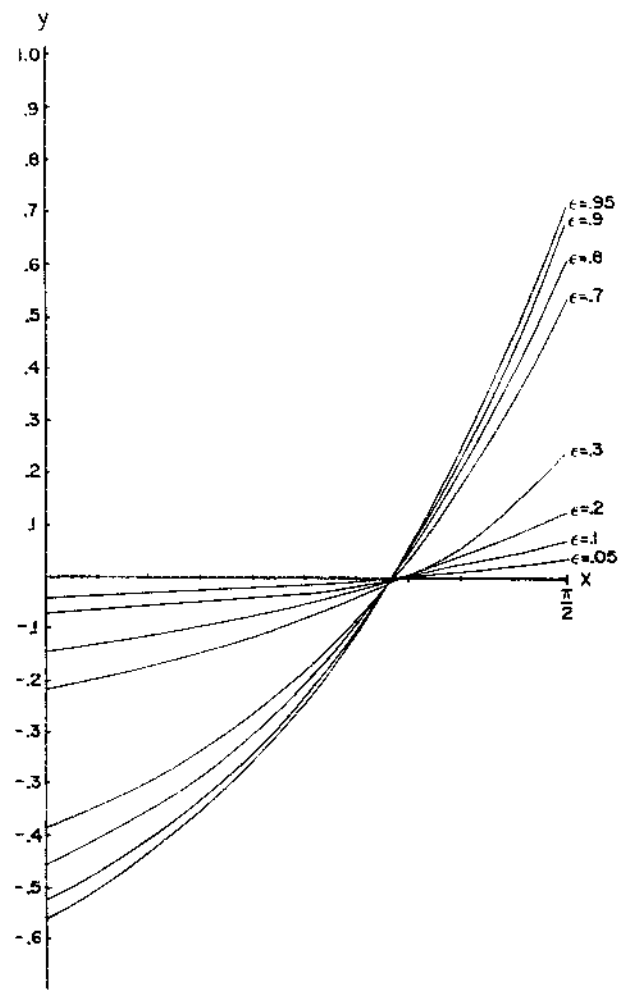

Fig. 4. Operator space for various ε (xsinx deleted from basis functions)

a larger ε gives a good approximation while performing poorly in the operator space.

REFERENCES

1 Birkhof, G. and Lynch, R. Numerical solution of elliptic problems, *SIAM Studies in Applied Math*, 1984
2 Hromadka II, T. V., Pinder, G. and Joos, B. Approximating a linear operator equation using a generalized Fourier series: Development, *Engineering Analysis*, 1987, 4(1)
3 Davis, P. J. and Rabinowitz, P. *Advances in Orthnormalizing Computation*. Academic Press, 1961
4 Hromadka II, T. V. and Yen, C. C. Complex boundary element model of flow field problems without matrices. *Engineering Analysis*, 1987, 4(1)
5 Kantorovich, L. V. and Krylov, V. I. *Approximate Methods of Higher Analysis*, Interscience Publishers, New York, 1964

APPENDIX A SOURCE PROGRAM – GENERALIZED FOURIER SERIES APPROXIMATION

```
PROGRAM EXAMPLE(GFSA_PAS, printer_LPT1);
(* y'' + y = 0     y(0)= 0     y(pi/2) = 1   *)

type
     index = integer;
     number = real;
     a40 = array[1..50] of REAL;
     a4040 = array[1..50,1..50] of REAL;
     a41 = array[0..50] of REAL;
var
     lst, data: text;
     x, y, b, s, value: a40;
     xk, g: a4040;
     node: a41;
     nbon, nbas, nint, ntol, kk, ntot1, i, j, k: index;
     epslo, sarea, ex, diff, sum, fbar, fd, xd, vbar, bk1, sum1: real;

     {$I b:GFSA.PAS}

procedure basis(var nb: index; nnod, i, kk: index;var xx, yy, v,
f: number; var xk, g: a4040; var node: a41);
     var
        x2, x3, y2, y3: real;
     begin
        x2 := sqr(xx);
        nb := 6;
        if kk=1 then
        begin
           v := xk[nb,1]*1 + xk[nb,2]*xx + xk[nb,3]*x2
                + xk[nb,4]*sin(xx) + xk[nb,5]*cos(xx) +
                  xk[nb,6]*xx*sin(xx)
                + xk[nb,7]*xx*cos(xx);

           f := xk[nb,1] + 2*xk[nb,3] +xk[nb,2]*xx + xk[nb,3]*x2
                + (xk[nb,6]*xx -2*xk[nb,5])*sin(xx)
                + (xk[nb,7]*xx + 2*xk[nb,4])*cos(xx);
        end
        else
            begin
                 if i > nnod then
                 begin
                   g[1,i] := 1;
                   g[2,i] := xx;
                   g[3,i] := 2 + x2;
                   g[4,i] := 0;
                   g[5,i] := 0;
                   g[6,i] := 2 * cos(xx);
                   g[7,i] := -2*sin(xx);
               end
               else
               begin
                    g[1,i] := 1;
                    g[2,i] := xx;
                    g[3,i] := x2;
                    g[4,i] := sin(xx);
                    g[5,i] := cos(xx);
                    g[6,i] := xx*sin(xx);
                    g[7,i] := xx*cos(xx);
```

```
                    end;
                  end;
    end; (* procedure basis *)

(* Main *)
begin

        assign(lst,'lpt1');
        rewrite(lst);
           writeln(lst,'  Generalized Fourier Analysis Program');

      nbon := 2;      (* The number of boundary points *)
      nint := 20;     (* The number of interior points *)
      epslo := 0.50;   (* Weighting for the boundary points *)


(* Compute the points and the functional*)

        x[1] := 0;
        x[2] := pi/2;
        y[1] := 0;
        y[2] := 0;
        value[1] := 0;
        value[2] := 1;
        ntol := nbon + nint;
        for i := (nbon+1) to ntol do
        begin
            x[i] := x[i-1] + 0.20;
            y[i] := 0;
            value[i] := 0;
        end;

(* Compute the area *)

       sarea := 0;

       for i := 1 to nbon do

       begin

            sarea := sarea + sqr(value[i]) * epslo;
       end;
       for i := (nbon+1) to ntol do
       begin
            sarea := sarea + sqr(value[i]) * (1-epslo);
     end;

(* Print the data points *)

            listdata;

(* Determine the Vector Coefficients *)

        for i := 1 to ntol do
        begin
             basis(nbas, nbon, i, 0, x[i], y[i], vbar, fbar, xk, g);

  end;
```

```
(* Modified Gram-Schmidt Procedure *)

        gram;

(* Approximate Boundary Point Values *)

        writeln(lst, '**  Approximate Boundary Point Errors*');
        writeln(lst,'Point':4,'Exact':20,'Approximation':20,
                'Rel Error':20);
         for i := 1 to nbon do
         begin
         basis(nbas, nbon, i, 1, x[i], y[i], vbar, fbar, xk,g);

         if value[i] = 0 then
                  xd := -9999
             else
             xd := (value[i]-vbar)/value[i];
             writeln(lst, i:4, value[i]:20, vbar:20, xd:20);
    end;

 (* Approximate Interior Point Values *)

         writeln(lst);
         writeln(lst, 'Approx. Interior Nodal Values&Errors');
         writeln(lst, 'Node':4,'Exact':20,'Approximation':20,'Rel Error':20
         for i := (nbon+1) to ntol do
         begin
             basis(nbas, nbon, i, 1, x[i], y[i], vbar, fbar,xk,g);
             ex := sin(x[i]);
             if ex <> 0 then
             xd := (ex-vbar)/ex
             else
               xd := -9999;
             writeln(lst, i:4, ex:20, vbar:20, xd:20);
         end;
         close(lst);
end.
(*  GP5A.PAS    Generalized Fourier Series Analysis Program *)
procedure listdata;
         begin
         assign(lst,'lpt1');
         rewrite(lst);

         writeln(lst,'Generalized Fourier Series Analysis Program');

         writeln(lst,'The number of boundary points = ',nbon);
         writeln(lst,'The number of interior points = ',nint);
         writeln(lst,'The weighting factor for the boundary points=',epslo);


(*  Print the Boundary Points  *)
         writeln(lst);
         writeln(lst, '***************    Boundary Nodes  ***************');


         writeln(lst, 'Node':4, 'X':20, 'Y':20, 'Value':20);

         for i := 1 to nbon do
         begin
             writeln(lst, i:4, x[i]:20, y[i]:20, value[i]:20);          end;
```

```
(* Print the interior points *)

        writeln(lst);
        writeln(lst,'*****************   Interior Points ****************');

  writeln(lst);

        ntol := nbon + nint;
        for i := (nbon + 1) to ntol do
        begin
             writeln(lst, i:4, x[i]:20, y[i]:20, value[i]:20);          end;
 end; (* procedure listdata *)

procedure gram;
  (* Modified Gram-Schmidt Orthogonalization Process *)
begin
        for i := 1 to nbas do
        begin
            sum := 0;
            for j := 1 to ntol do
            begin
                if j <= nbon then
                   sum := sum + epslo * sqr(g[i,j])
                else
                   sum := sum + (1 - epslo) * sqr(g[i,j]);
     end;

            sum := sqrt(sum);
            s[i] := sum;

            if sum < 0.000001 then
               s[i] := 0;

            for j := 1 to ntol do
            begin
                if sum > 0 then
                   g[i,j] := g[i,j]/sum
            else
               g[i,j] := 0;
        end;

        if i <> nbas then
        begin
            for kk := (i + 1) to nbas do
            begin
                suml := 0;
                for j := 1 to ntol do
                begin
                    if j > nbon then
                      suml := suml + (1-epslo) * g[kk,j] *g[i,j]

       else
                      suml := suml + g[i,j] * g[kk,j] *epslo;

 end;

                if s[i] > 0 then
                   xk[kk,i] :=  -1 * suml / s[i]
                else
                   xk[kk,i] := 0;
```

```
                    for j := 1 to ntol do
                    begin
                        g[kk,j] := g[kk,j] - (sum1 * g[i,j]);
                end;

                end;
            end;
        end;


      writeln(lst,'*************   Orthogonality Test ***********');
        for i := 1 to (nbas-1) do
      begin
          for k := (i+1) to nbas do
          begin
              sum := 0;
              for j := 1 to ntol do
              begin
                  if j <= nbon then
                      sum := sum + g[i,j] * g[k,j] * epslo
           else
                      sum := sum + g[i,j] * g[k,j] * (1-epslo);
            end;
            (*  writeln(lst, i:4, k:4, sum:20); *)
          end;
        end;

(* Compute the Coefficients of B[i] = (Value[i], G[i]) *)

        writeln(lst,'****  Evaluation Coefficients   G[i].B[i] /G[i].G[i] ');

        sum := 0;
        for i := 1 to nbas do
            begin
                bk1 := 0;
                for j := 1 to ntol do
                begin
                    if j <= nbon then
                        bk1 := bk1 + value[j] * g[i,j] * epslo
               else
                        bk1 := bk1 + value[j] * g[i,j] * (1-epslo);
                end;

(* Compute the Norm of the Generalized Fourier Coefficients *)

                if s[i] > 0.0000001 then
                    b[i] := bk1/s[i]
                else
                    b[i] := 0;

                sum := sum + sqr(bk1);
            end;

            for i := 1 to nbas do
                writeln(lst,'b[',i:4,'] ', b[i]:20);

            writeln(lst);
```

```
(* Compute Basis Function Coefficients by Back Substitution *)

        for i := nbas downto 1 do
        begin
            if i = nbas then
               xk[nbas,i] := b[nbas]
            else
               xk[nbas,i] := xk[nbas,i] * b[nbas] +b[i];
        end;

        for i := (nbas-1) downto 1 do
        begin
            for j := i downto 1 do
            begin
                if i<>j then
                   xk[nbas,j] := xk[nbas,i] * xk[i,j]+xk[nbas,j];

            end;
        end;

        writeln(lst, '  ****   Back Substitution Coefficients ****');
writeln(lst);

        for i := 1 to nbas do
            writeln(lst, xk[nbas,i]:20);

        writeln(lst);
     end;
```