

```

Aug 11, 07 21:57          fem.adb          Page 1/4
-- Ada implementation of the simple finite elements program
-- for Mathematics 503, CSUF summer 2007.
--
-- by Nasser Abbasi
--
-- environment: GNAT 2007 on Linux. Uses Ada 2005 functions.
--
-- date: August 10, 2007.
--
-- compile command:
-- gnatmake fem.adb -largS -L/usr/lib -lgnala -llapack -lblas
--
-- run command:
-- fem > out.txt
--
-- plot the result using gnuplot as follows:
-- gnuplot
-- >plot "out.txt"
--
-- To DO:
-- This implementation in Ada was just for fun, proof of concept
-- to try the new Ada 2005 linear algebra functions. This
-- Finite Elements code is not efficient. I use NxN matrix
-- even though the A matrix is only tridiagonal. Also a
-- tridiagonal solver is needed which Ada 2005 does not have
-- in its library, but easy to write.
--
-- I wrote this same code in Mathematica as well, see my
-- project web site.
--
-- Overall, I was pleased with the Ada implementation. It took
-- less time than I expected to implement. only 1 hour, and
-- the Ada compiler caught all my errors at compile time, and
-- I did not have any errors at run time. Also with Ada, I was
-- able to define arrays that start at 0 and not 1, this allowed
-- me an easier implementation.
--
-- For plotting, I print the output array (data points, (x,y)) and
-- from the command line direct the output of the program to a
-- textfile, then one can use Matlab or Mathematica or gnuplot or
-- any other program to display the output.
--
-- This code can be more modular and can use more data hiding,
-- may be use a package and make some of these functions more
-- reusable. But for now, I achieved my goal or trying Ada for
-- this, and I found that Ada makes great language for something
-- like this.

with Ada.Text_IO, Ada.Integer_Text_IO, Ada.Float_Text_IO;
use Ada.Text_IO, Ada.Integer_Text_IO, Ada.Float_Text_IO;
with Ada.Numerics.Generic_Real_Arrays;

procedure fem is

  package myNumerics is new Ada.Numerics.Generic_Real_Arrays (Float);
  use myNumerics;

  nElements      : constant Positive := 200;
  nShapeFunctions : constant Positive := nElements - 1;
  nPoints        : constant Positive := nElements + 1;

```

```

Aug 11, 07 21:57          fem.adb          Page 2/4
A      : Real_Matrix (1 .. nShapeFunctions, 1 .. nShapeFunctions)
:= (others => (others => 0.0));

c, b   : Real_Vector (1 .. nShapeFunctions) := (others => 0.0);

data_points : Real_Matrix (0 .. nShapeFunctions + 1, 1 .. 2) :=
  (others => (others => 0.0));

q      : constant Float := 4.0;
f      : constant Float := 4.0;
L      : constant Float := 1.0;
h      : Float := L / Float (nElements);
Left_Boundary : constant Float := 0.0;
Right_Boundary : constant Float := 0.0;

--
-- function zeta
--
function zeta (x : in Float; L : in Float) return Float is
begin

  if x >= 0.0 and x <= L then
    return 1.0 - x;
  else
    if x > L then
      return 0.0;
    else
      return zeta (-x, L);
    end if;
  end if;

end zeta;

--
-- function phi
--
function phi
(x : in Float;
 i : in Positive;
 h : in Float;
 L : in Float)
return Float
is
begin
  return zeta (x / h - Float (i), L);
end phi;

--
-- procedure Fill_Tridiagonal_Matrix
--
procedure Fill_Tridiagonal_Matrix (A : in out Real_Matrix) is
begin

  for i in A'Range (1) loop
    for j in A'Range (2) loop
      if i = j then
        A (i, j) := 2.0 + (2.0 / 3.0) * q * h ** 2;
      else
        if j = i - 1 or j = i + 1 then
          A (i, j) := -1.0 + (1.0 / 6.0) * q * h ** 2;
        end if;
      end if;
    end loop;
  end loop;

```

Aug 11, 07 21:57

fem.adb

Page 3/4

```

    end loop;
  end loop;

end Fill_Tridiagonal_Matrix;

--
-- Fill_Vector
--
procedure Fill_Vector (b : in out Real_Vector) is
begin

  for i in b'Range (1) loop
    b (i) := f * h ** 2;
  end loop;

end Fill_Vector;

--
-- Print_Matrix
--
procedure Print_Matrix (A : in Real_Matrix) is
begin

  for i in A'Range (1) loop
    for j in A'Range (2) loop
      Put (A (i, j));
      Put (" ");
    end loop;
    New_Line;
  end loop;

end Print_Matrix;

--
-- yApprox
--
function yApprox
(c : in Real_Vector;
x : in Float;
h : in Float;
L : in Float)
return Float
is
  sum : Float := 0.0;
begin
  for i in c'Range loop
    sum := sum + c (i) * phi (x, i, h, L);
  end loop;
  return sum;
end yApprox;

--
-- set_boundary_conditions
--
procedure Set_Boundary_Conditions is
begin
  data_points (data_points'First (1), 1) := 0.0;
  data_points (data_points'First (1), 2) := Left_Boundary;
  data_points (data_points'Last (1), 1) := L;
  data_points (data_points'Last (1), 2) := Right_Boundary;
end Set_Boundary_Conditions;

```

Aug 11, 07 21:57

fem.adb

Page 4/4

```

begin

  --Put_Line("Welcome small Finite Elements Program!");
  Fill_Tridiagonal_Matrix (A);
  Fill_Vector (b);
  --print_matrix(A);
  c := Solve (A, b);

  Set_Boundary_Conditions;

  for i in 1 .. nShapeFunctions loop
    data_points (i, 1) := Float (i) * h;
    data_points (i, 2) := yApprox (c, data_points (i, 1), h, L);
  end loop;

  Print_Matrix (data_points);

end fem;

```