In [3]:

```
Autosave disabled
```

In [1]:

```
Collecting scikit-fem
  Downloading scikit_fem-8.0.0-py3-none-any.whl (157 kB)
     ---------------------------------------- 157.8/157.8 kB 3.1 MB/s eta 0:00:00
Requirement already satisfied: scipy in c:\users\gary\appdata\local\programs\python\python39\lib\si
te-packages (from scikit-fem) (1.9.3)
Requirement already satisfied: numpy in c:\users\gary\appdata\local\programs\python\python39\lib\si
te-packages (from scikit-fem) (1.23.5)
Installing collected packages: scikit-fem
Successfully installed scikit-fem-8.0.0
```

Chapter 31-15 Solving PDEs with the Finite Element Method.

The finite element method (FEM) is a popular method for numerically solving differential equations arising in engineering and mathematical modeling. Typical problem areas of interest include the traditional fields of structural analysis, heat transfer, fluid flow, mass transport, and electromagnetic potential.

The FEM is a general numerical method for solving partial differential equations in two or three space variables (i.e., some boundary value problems). To solve a problem, the FEM subdivides a large system into smaller, simpler parts that are called finite elements. This is achieved by a particular space discretization in the space dimensions, which is implemented by the construction of a mesh of the object: the numerical domain for the solution, which has a finite number of points. The finite element method formulation of a boundary value problem finally results in a system of algebraic equations. The method approximates the unknown function over the domain. The simple equations that model these finite elements are then assembled into a larger system of equations that models the entire problem. The FEM then approximates a solution by minimizing an associated error function via the calculus of variations.

from Wikipedia

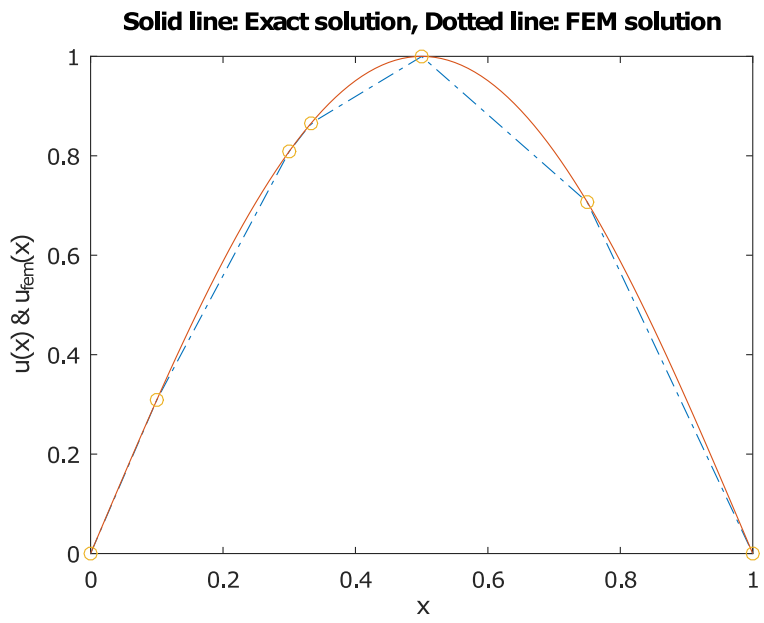
1. This problem depends on material from Chapters 6 and 8 of the book *Numerical Solution of Differential Equations -- Introduction to Finite Difference and Finite Element Methods* by Li, Qiao, and Tang. At the address *https://zhilin.math.ncsu.edu/ (https://zhilin.math.ncsu.edu/)* is Li's page featuring details about the book. There is a button on the page, just underneath the book title, with the label "Matlab Codes". The download will be entitled "DE_FEM_Matlab_Code.zip" All of the files in the folder with the title "Chapter 6" should be loaded into Octave. Only one will execute in Octave, the one entitled drive.m. This file is shown below.

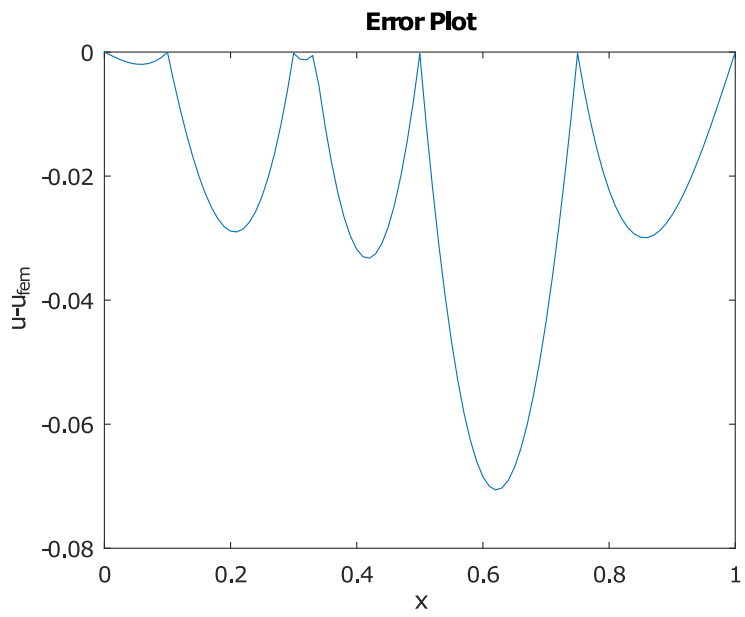
In [ ]:

```
clear all; close all;   % Clear every thing so it won't mess up with other
                        % existing variables.

%%%%%% Generate a triangulation

 x(1)=0; x(2)=0.1; x(3)=0.3; x(4)=0.333; x(5)=0.5; x(6)=0.75;x(7)=1;

%x=0:0.05:1;

%%%%%% Call fem1d function to get the FEM solution at nodal points.

U = fem1d(x);

%%%%%% Compare errors:

x2 = 0:0.01:1; k2 = length(x2);
for i=1:k2,
  u_exact(i) = soln(x2(i));
  u_fem(i) = fem_soln(x,U,x2(i));  % Compute FEM solution at x2(i)
end

error = norm(u_fem-u_exact,inf)  % Compute the infinity error

plot(x2,u_fem,'-.', x2,u_exact)   % Solid: the exact, %dotted: FEM solution
hold; plot(x,U,'o')              % Mark the solution at nodal points.
% set(gca,'FontSize',18);
xlabel('x'); ylabel('u(x) & u_{fem}(x)');
title('Solid line: Exact solution, Dotted line: FEM solution')

figure(2); % set(gca,'FontSize',18);
plot(x2,u_fem-u_exact); title('Error plot')
xlabel('x'); ylabel('u-u_{fem}');  title('Error Plot')

```

If access to the book mentioned above is possible, some good info regarding the relation between PDEs and finite element modeling can be learned.

The plots which are produced by running the code in the above cell are shown below.





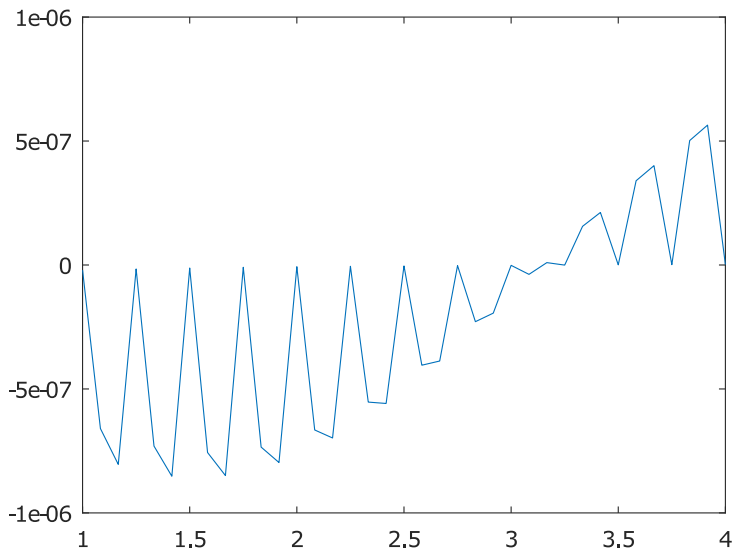
3. For the initial conditions shown in the comment banner in the cell below, construct a 1-dimensional finite element solution and plot.

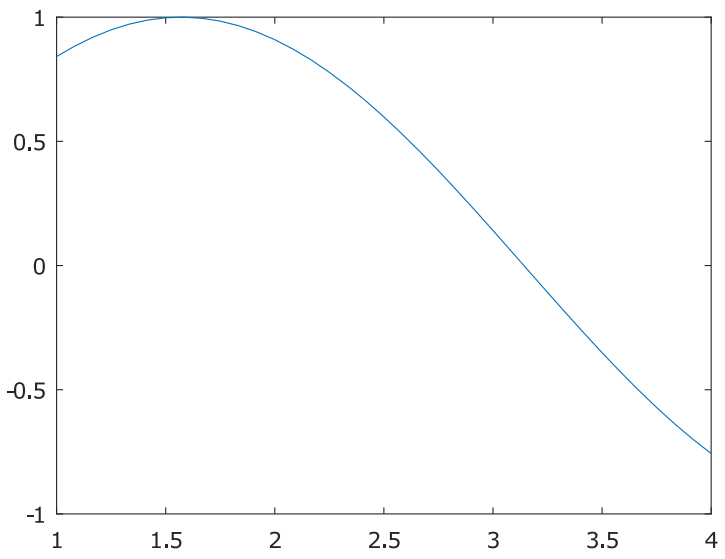
The plot below on the left shows the error; the plot on the right shows the analytical solution.

In [ ]:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  Matlab program for one-dimensinal finite element method for                %
%                                                                             %
%    -(k(x)u_x)_x + c(x) u_x + b(x) u(x) = f(x)                               %
%                                                                             %
%  with different boundary condition at x =0, and x = l                       %
%                                                                             %
%-----------------------------------------------------------------------------%

    function [x,u]= fem1d

   clear
%  Preprocessor:

   global nnode nelem
   global gk gf
   global xi  w

%%%%%%%%% Start the program %%%%%%%%%

%   clear

   [xi,w] = setint;   % Get Gaussian points and weights.

%  Preprocessor:
   [x,kbc,vbc,kind,nint,nodes] = propset;         % Input data

    formkf(kind,nint,nodes,x,xi,w);   % Form the discrete system

    aplybc(kbc,vbc);

    u = gk\gf;                   % Solve the linear system.

    for i=1:nnode,
      e(i) = u(i) - uexact(x(i));
    end
    figure(1); plot(x,e)
    figure(2); plot(x,u)
```

```
51
```





In [ ]:

The FEnics package requires Dolfin as a component, and Dolfin can be hard to install. NGSolve requires the MKL library, which is a big download if the CPU is not Intel. Some other finite element packages require Petsc, which can be difficult to get running, even on Linux.

In Contrast, the Python module scikit-fem is a lightweight but sturdy implement for finite element calculations in Python. It does not need compiling, and also entails no dependencies for the average Python installation. The next few cells below contain the first exercise in the Getting Started section of the scikit-fem documentation.

4. Solve the Poisson problem

$$-\Delta u = f \qquad \text{in } \Omega$$

$$u = 0 \qquad \text{on } \partial\Omega$$

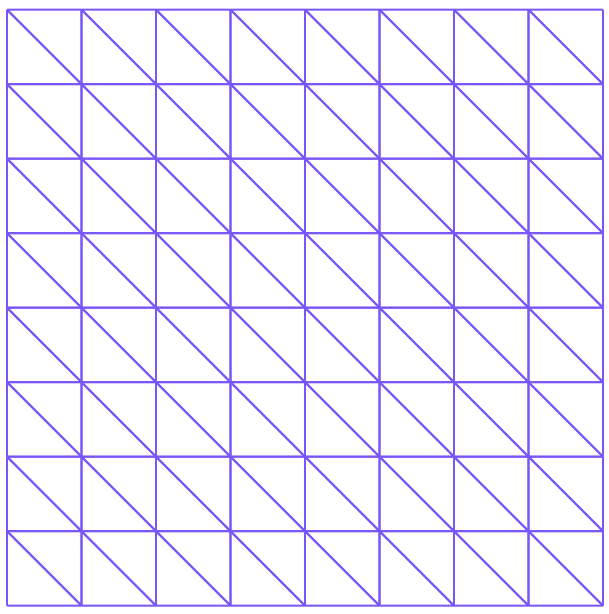
where $\Omega = (0,1)^2$ is a square domain and $f(x,y) = \sin \pi x \sin \pi y$

In [1]:
```
import skfem as fem
>>> from skfem.helpers import dot, grad  # helpers make forms look nice
>>> @fem.BilinearForm
... def a(u, v, _):
...     return dot(grad(u), grad(v))
```

In [2]:
```
import numpy as np
>>> @fem.LinearForm
... def L(v, w):
...     x, y = w.x  # global coordinates
...     f = np.sin(np.pi * x) * np.sin(np.pi * y)
...     return f * v
```

In [4]:
```
mesh = fem.MeshTri().refined(3)  # refine thrice
mesh
```

Out[4]:



In [6]:
```
Vh = fem.Basis(mesh, fem.ElementTriP1())
Vh
```

Out[6]:
```
<skfem CellBasis(MeshTri1, ElementTriP1) object>
  Number of elements: 128
  Number of DOFs: 81
  Size: 27648 B
```

In [9]:
```
A = a.assemble(Vh)
l = L.assemble(Vh)
A.shape

```

Out[9]: (81, 81)

In [10]:
```
l.shape

```

Out[10]: (81,)

In [11]:
```
D = Vh.get_dofs()
D

```

Out[11]:
```
<skfem DofsView(MeshTri1, ElementTriP1) object>
  Number of nodal DOFs: 32 ['u']
```

In [13]:
```
x = fem.solve(*fem.condense(A, l, D=D))
x.shape

```

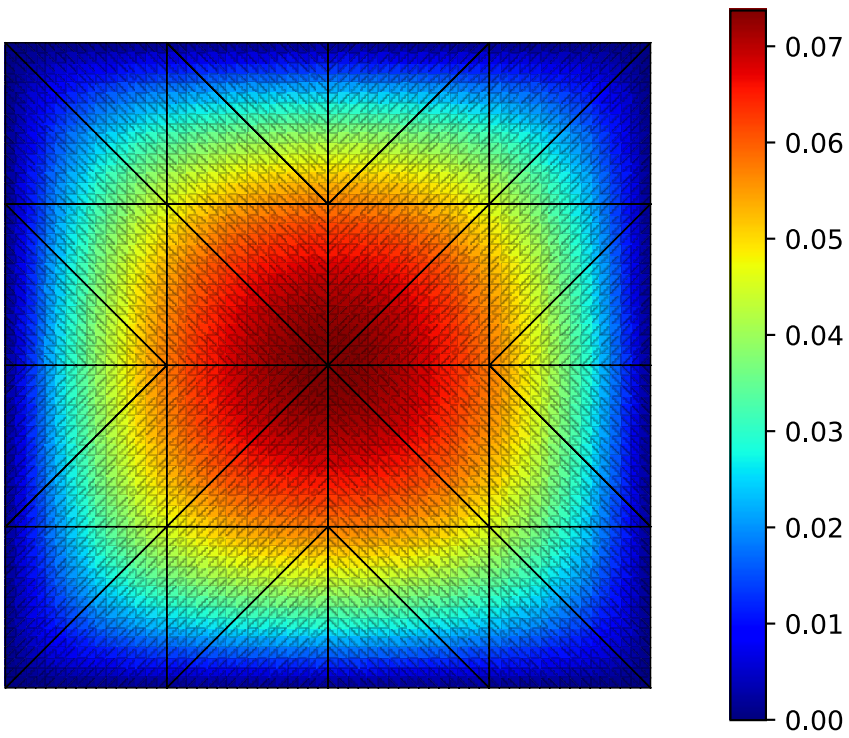
Out[13]: (81,)

In [15]:
```
@fem.Functional
def error(w):
    x, y = w.x
    uh = w['uh']
    u = np.sin(np.pi * x) * np.sin(np.pi * y) / (2. * np.pi ** 2)
    return (uh - u) ** 2
round(error.assemble(Vh, uh=Vh.interpolate(x)), 9)

```

Out[15]: 1.069e-06

In [17]:

```
"""Discontinuous Galerkin method."""

from skfem import *
from skfem.helpers import grad, dot, jump
from skfem.models.poisson import laplace, unit_load

m = MeshTri.init_sqsymmetric().refined()
e = ElementTriDG(ElementTriP4())
alpha = 1e-3

ib = Basis(m, e)
bb = FacetBasis(m, e)
fb = [InteriorFacetBasis(m, e, side=i) for i in [0, 1]]

@BilinearForm
def dgform(u, v, p):
    ju, jv = jump(p, u, v)
    h = p.h
    n = p.n
    return ju * jv / (alpha * h) - dot(grad(u), n) * jv - dot(grad(v), n) * ju

@BilinearForm
def nitscheform(u, v, p):
    h = p.h
    n = p.n
    return u * v / (alpha * h) - dot(grad(u), n) * v - dot(grad(v), n) * u

A = asm(laplace, ib)
B = asm(dgform, fb, fb)
C = asm(nitscheform, bb)
b = asm(unit_load, ib)

x = solve(A + B + C, b)

M, X = ib.refinterp(x, 4)

def visualize():
    from skfem.visuals.matplotlib import plot, draw
    %config InlineBackend.figure_formats = ['svg']

    ax = draw(M, boundaries_only=True)
    return plot(M, X, shading="gouraud", ax=ax, colorbar=True)

if __name__ == "__main__":
    visualize().show()

```



The Github repository of *gdmcbain/fenics-tuto-in-skfem* contains eight exercises from the FEnics tutorial book. FEnics's great popularity no doubts stems largely from the book by Langtangen and Logg, entitled *Solving PDEs in Python: The FEnics Tutorial I*, a full-length book about using FEnics for finite element solutions of PDEs. This book can be accessed on line or downloaded for free. Since FEnics is such a popular package, it is natural for someone to wonder if the new, little-known and somewhat brash program scikit-fem could reproduce the results of the FEnics tutorials. Below are six of the eight problems of the collection. (Note: problems 2 and 6 could not be attempted because they require a commercial program for execution.)