Solving Convection - Diffusion Equation using Finite Element Methods and its stabilization techniques

Problem 1:

Consider the 1d convection-diffusion equation with Dirichlet boundary conditions :

$$-\epsilon u'' + bu' = 1, \quad \text{in} \quad \Omega = [0,1]$$
$$u(0) = u(1) = 0$$

where, ϵ and b are diffusion and convection coefficients respectively. We seek to find the solution $u \in V$, where V is some finite element space satisfying the boundary condition. Note that the source term, i.e the r.h.s of the equation is constant (1 here).

The exact solution to the above problem is

$$u = \frac{1}{b}\left(x - \frac{1 - e^{\frac{bx}{\epsilon}}}{1 - e^{\frac{b}{\epsilon}}}\right)$$

1. Galerkin

We write the above problem in variational form by multiplying by test function $v \in V$

$$\int_\Omega (-\epsilon u'' + bu')v\,dx = \int_\Omega v\,dx$$

Integrating by parts over domain Ω

$$\epsilon \int_\Omega u'v'\,dx + b\int_\Omega u'v\,dx = \int_\Omega v\,dx$$

Now let us write the problem in compact form as follows

$$u \in V: \quad a(u,v) = l(v) \qquad \forall v \in V$$
$$a(u,v) := \epsilon \int_\Omega u'v'\,dx + b\int_\Omega u'v\,dx$$
$$l(v) := \int_\Omega v\,dx$$

The above problem is solved below by using `firedrake` .

Import the required libraries i.e `firedrake` , `matplotlib` and `numpy`

In [1]:

```
import firedrake as fe
import matplotlib.pyplot as plt
import numpy as np
```

The mesh for the above 1d problem is defined using `firedrake.UnitIntervalMesh` . Here we specify N, the number of elements in the mesh as 10.

In [2]:

```
N = 10
mesh = fe.UnitIntervalMesh(N)
```

Define the value of diffusion coefficient ϵ and convection coefficient b for the problem.

In [3]:

```
epsilon = 1  # diffusion constant
b = 1 # convection constant
```

The Mesh Peclet number is defined as

$$Pe = \frac{bh}{2\epsilon}$$

where h is the mesh cell size. Here, $h = \frac{1}{N}$.

In [4]:

```
def peclet_number(b, N, epsilon):
    """ returns Peclet number

    Keyword Arguments:

    b - convection coefficient

    N - mesh size

    epsilon - diffusion coefficient
    """
    h = 1./N

    return b*h/(2.*epsilon)
```

In [5]:

```
pe = peclet_number(b,N,epsilon)
print('Pe =',pe)
```

```
Pe = 0.05
```

Now, we shall define the finite dimensional function space V in which the problem is to be solved. It is defined using `firedrake.FunctionSpace` which takes the mesh, the finite element family and degree of the finite element family as inputs. We shall choose the space of piecewise linear polynomials that are continuous between elements.

In [6]:

```
V = fe.FunctionSpace(mesh, 'CG', 1)
```

Now, we shall define the `TestFunction` and `Trial function` on the function space V

In [7]:

```
u = fe.TrialFunction(V)
v = fe.TestFunction(V)
```

Let us get the `SpatialCoordinates` x for the mesh.

In [8]:

```
x = fe.SpatialCoordinate(mesh)[0]
```

Let us now define the r.h.s of the equation . Since, the r.h.s is a constant (1 here), we shall use `firedrake.Constant` .

In [9]:

```
f = fe.Constant(1)
```

Now we shall impose the Dirichlet Boundary condition for the problem. We need to first define the boundary IDs. To determine the boundary IDs we call help for `UnitIntervalMesh` as follows.

In [10]:

```
help(fe.UnitIntervalMesh)
```

```
Help on function UnitIntervalMesh in module firedrake.utility_meshes:

UnitIntervalMesh(ncells, distribution_parameters=None, comm=<mpi4py.MPI.In
tracomm object at 0x7fff2e7a0060>)
    Generate a uniform mesh of the interval [0,1].

    :arg ncells: The number of the cells over the interval.
    :kwarg comm: Optional communicator to build the mesh on (defaults to
        COMM_WORLD).

    The left hand (:math:`x=0`) boundary point has boundary marker 1,
    while the right hand (:math:`x=1`) point has marker 2.
```

For `UnitIntervalMesh` the boundary IDs 1 and 2 correspond to $x = 0$ and $x = 1$ respectively. Once we define the boundary IDs we impose the Dirichlet boundary conditions by using `firedrake.DirichletBC` by passing function space V, the value at boundary (0 here) and boundary ids as arguments.

In [11]:

```
value = 0
boundary_ids = {1,2} # boundaries x=0 and x=1
bc = fe.DirichletBC(V,value,boundary_ids) # Boundary conditions for x=0 and x=1, u=0
```

Let us define $u_h \in V$ holding the solution. We shall do that by declaring u_h as a `Function` in V.

In [12]:

```
uh = fe.Function(V)
```

Let us now define the linear and bilinear form as follows.

$$u \in V: \quad a(u,v) = l(v) \qquad \forall v \in V$$
$$a(u,v) := \epsilon \int_\Omega u'v'\,dx + b\int_\Omega u'v\,dx$$
$$l(v) := \int_\Omega v\,dx$$

In [13]:

```
dx, grad, dot = fe.dx, fe.grad, fe.dot

L = f*v*dx
a =  ( epsilon * dot(grad(v),grad(u)) ) * dx + b * u.dx(0) * v * dx
```

Now we shall solve the problem using `firedrake.solve` .

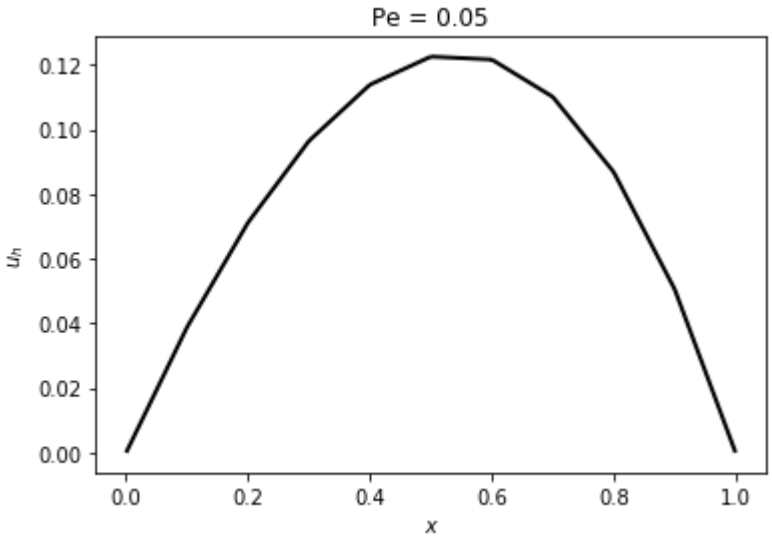
In [14]:

```
fe.solve(a == L, uh, bcs=bc)
```

Now we shall plot the results using the `firedrake.plot` function.

In [15]:

```
fe.plot(uh)
plt.title('Pe = %.2f'%pe)
plt.xlabel('$x$')
_ = plt.ylabel('$u_h$')
```



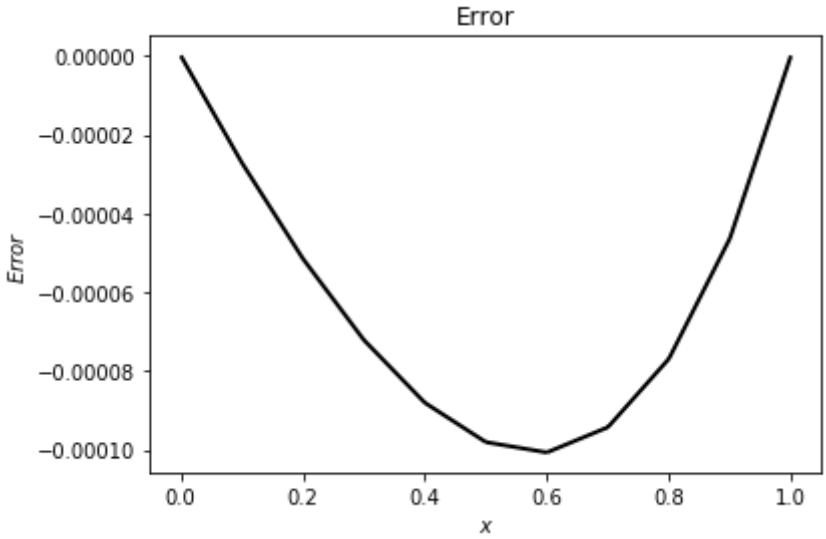
Since we have the exact solution to the problem, we shall look at the error produced.

The exact solution to the above problem is given by

$$u = \frac{1}{b}\left(x - \frac{1 - e^{\frac{bx}{\epsilon}}}{1 - e^{\frac{b}{\epsilon}}}\right)$$

In [16]:

```
u_exact = (x - (1-fe.exp(b*x/epsilon))/(1-fe.exp(b/epsilon)))/b
difference = fe.assemble(fe.interpolate(u_exact, V) - uh)
fe.plot(difference)
plt.xlabel('$x$')
plt.ylabel('$Error$')
_ = plt.title('Error')
```



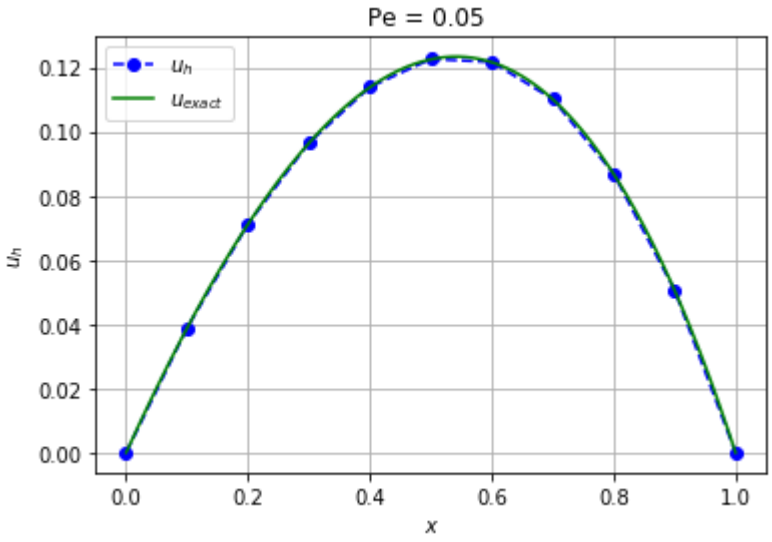
Since the 1d plotting interface of `firedrake` is a bit difficult to visualize the numerical solution and the exact solution together, we shall use the `matplotlib.pyplot` to plot. We define a function `myplot` which takes numerical solution u_h, diffusion coefficient b, convection coefficient ϵ and Peclet number Pe as arguments and plots the exact and numerical solution.

In [17]:

```
def myplot(uh, b, epsilon, pe):
    """Plots 1d solution and exact solution

    Key arguments:

    uh - Numerical solution to the 1-d problem

    b - convection coefficient

    epsilon - diffusion coefficient

    pe - Peclet number

    """
    x = np.arange(0,1.0001,0.1)


    plt.plot(x,uh(x),'--ob', label = '$u_h$')

    x = np.arange(0,1.0001,0.0001)
    u_exact = (x - (1-np.exp(b*x/epsilon))/(1-np.exp(b/epsilon)))/b

    plt.plot(x,u_exact,'-g', label = '$u_{exact}$')
    plt.xlabel('$x$')
    plt.ylabel('$u_h$')
    plt.title('Pe = %.2f '%pe)
    plt.legend(loc = 'upper left')
    plt.grid(True)

    return
```

In [19]:

```
myplot(uh,b,epsilon,pe)
```



Now, let us write the code as a function which solves the above problem by Galerkin by taking convection coefficient ϵ, diffusion coefficient b and mesh size N as arguments and plots the approximate solution and exact solution.

In [20]:

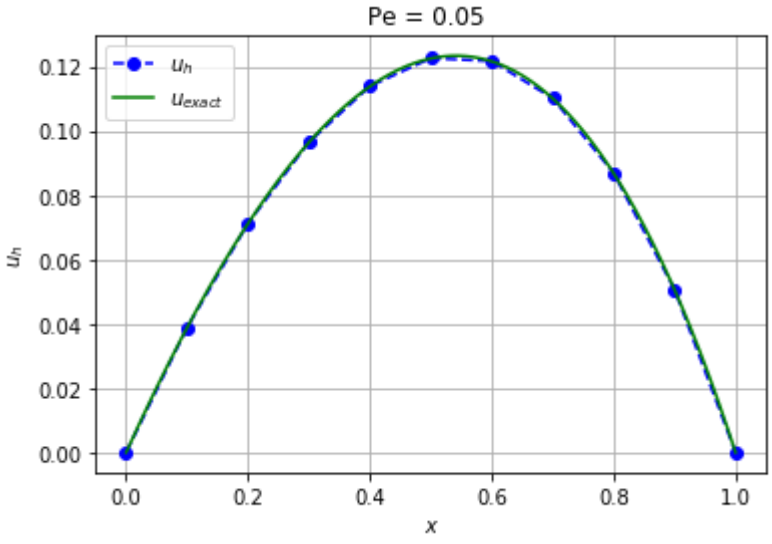
```
def galerkin(epsilon, b, N):
    """Solves the 1-d problem by Galerin

    Key Arguments:

    epsilon - diffusion coefficient

    b -  convection coefficient

    N - mesh size

    """
    e = epsilon

    h = 1/N

    pe = peclet_number(b,N,e)
    print('Pe =',pe)

    L = f*v*dx
    a =  ( e* dot(grad(v),grad(u)) ) * dx + b * u.dx(0) * v * dx

    fe.solve(a == L, uh, bcs=bc)

    myplot(uh, b, e, pe)

    return
```

Now we shall solve the problem for different values of diffusion coefficient ϵ and convection coeffient b and compare the analytical and exact solution for the different values of Peclet number Pe.

In [21]:

```
galerkin(epsilon=1,b=1,N=10)
```

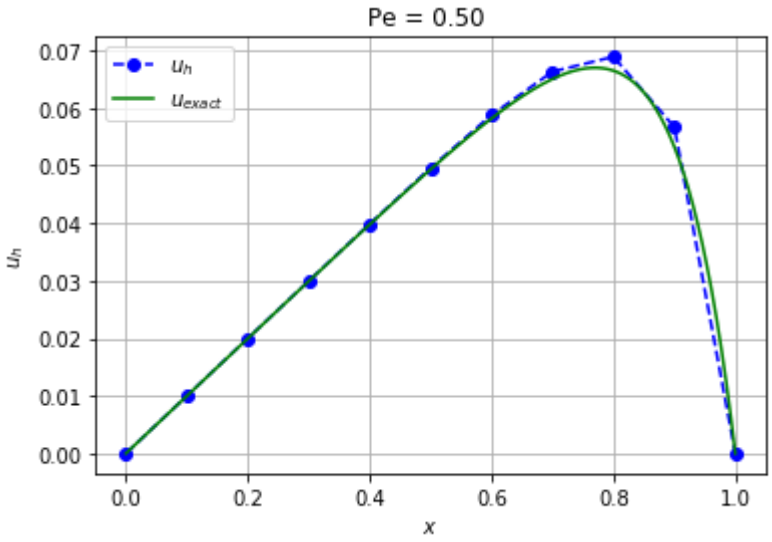
```
Pe = 0.05
```



In [22]:

```
galerkin(epsilon=1,b=10,N=10)
```

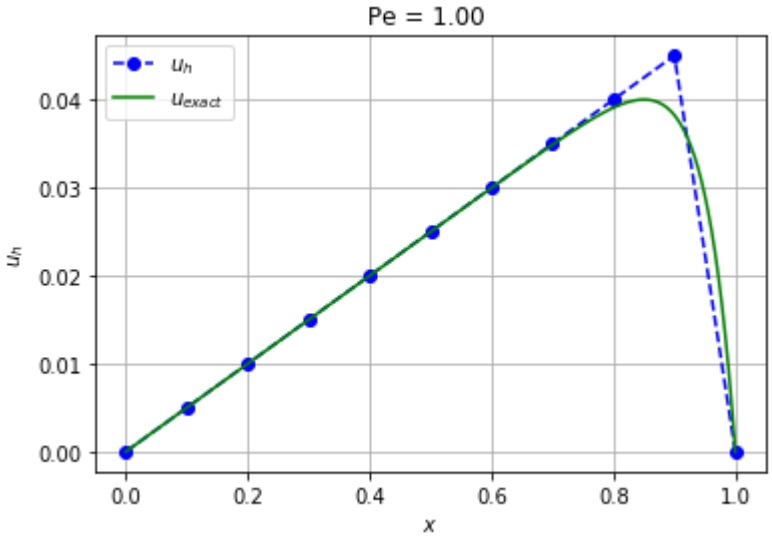
```
Pe = 0.5
```



In [23]:

```
galerkin(epsilon=1,b=20,N=10)
```

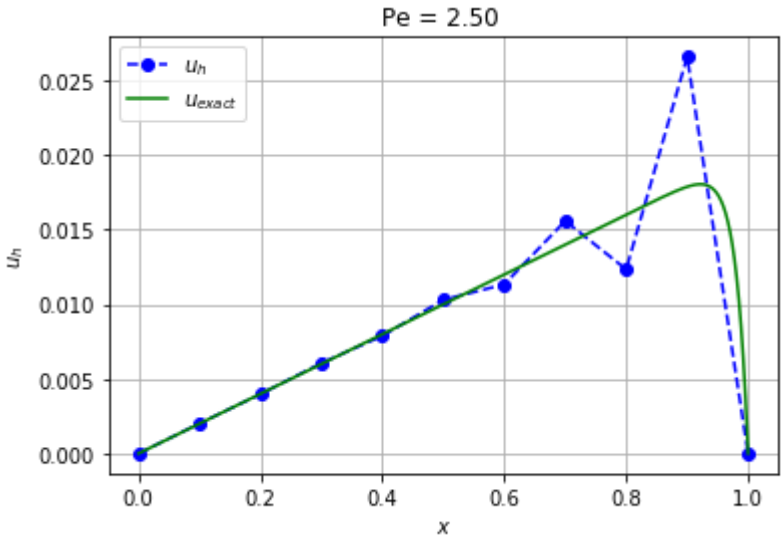
```
Pe = 1.0
```



In [24]:

```
galerkin(epsilon=1,b=50,N=10)
```

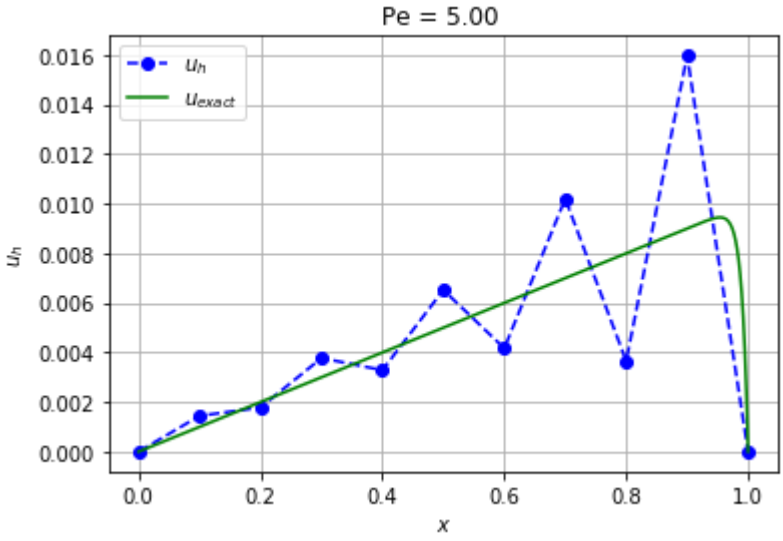
Pe = 2.5



In [25]:

```
galerkin(epsilon=1,b=100,N=10)
```

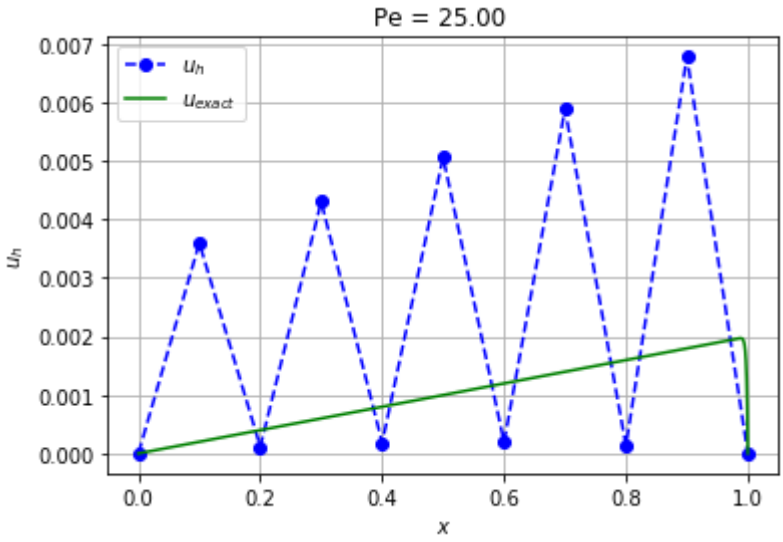
Pe = 5.0



In [26]:

```
galerkin(epsilon=1,b=500,N=10)
```

```
Pe = 25.0
```



It can be seen that the Galerkin method has problems in the convection dominated cases $Pe > 1$ where the solution is oscillatory. A way of overcoming this is by increasing the number of points in the grid to force $Pe \approx 1$. However, this is not always possible as it increases computatonal cost significantly.

Following are some ways to stabilize the convection-diffusion equation.

2. Artificial Diffusion

In this method, an artificial diffusion is added such that it reduces the mesh Peclet number to an effective Peclet number of almost unity.

Now, the variational form becomes

$$(\epsilon + \epsilon_h) \int_\Omega u'v'\,dx \; + b \int_\Omega u'v\,dx = \int_\Omega v\,dx$$

where, ϵ_h is the artificial diffusion coefficient.

Now let us rewrite the problem in compact form as follows

$$u \in V: \qquad a(u,v) = l(v) \qquad \forall v \in V$$
$$a(u,v) := (\epsilon + \epsilon_h) \int_\Omega u'v'\,dx \; + b \int_\Omega u'v\,dx$$
$$l(v) := \int_\Omega v\,dx$$

The value of ϵ_h is chosen such that it reduces the Peclet number. Choose $\epsilon_h = \frac{\gamma}{2} hb$

If $\gamma \geq 1$, the effective Peclet number is sufficiently small and hence diffusion is minimized.

For perfect stabilization we can use $\gamma = \coth(Pe) - 1/Pe$.

Let us write a function `artificial_diffusion` to solve the problem by adding artficial diffusion ϵ_h and plots the numerical approximation against the exact solution.

In [27]:

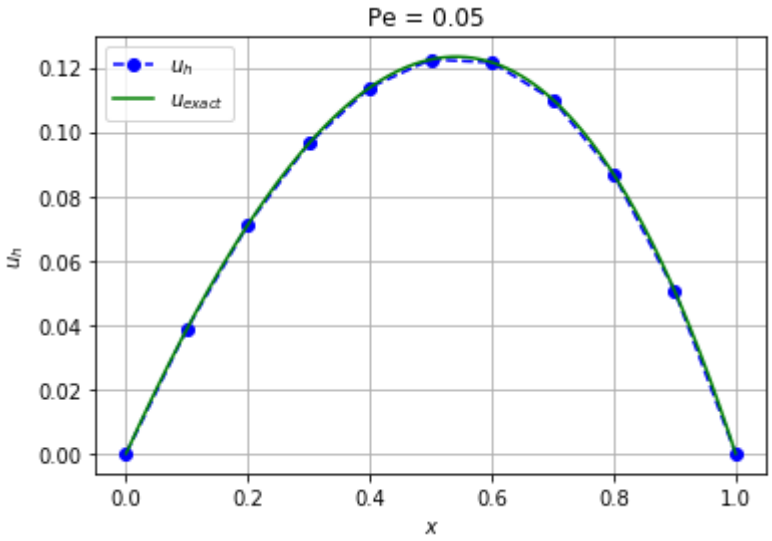
```
def artificial_diffusion(epsilon, b, N):
    """Solves the 1-d problem by Artficial diffusion

    Key Arguments:

    epsilon - diffusion coefficient

    b - convection coefficient

    N - mesh size

    """
    e = epsilon

    h = 1/N

    pe = peclet_number(b,N,e)
    print('Pe =',pe)

    gamma = ( ((fe.exp(2*pe)+1)/(fe.exp(2*pe)-1)) - 1/pe)  # For perfect stabilization
    eh = gamma/2*h*b # artificial diffusion coefficient
    pe_eff = b*h/(2*(e+eh)) # Effective Peclet number
    print('Effective Pe = ',pe_eff)

    dx, dot, grad = fe.dx, fe.dot, fe.grad
    L = f*v*dx
    a =  ( (e+eh)* dot(grad(v),grad(u)) ) * dx + b * u.dx(0) * v * dx

    fe.solve(a == L, uh, bcs=bc)

    myplot(uh, b, e, pe)

    return
```

Now we shall solve the problem by artificial diffusion method for different values of diffusion coefficient ϵ and convection coeffient b and compare the analytical and exact solution for the different values of Peclet number Pe.

In [28]:

```
artificial_diffusion(epsilon=1,b=1,N=10)
```

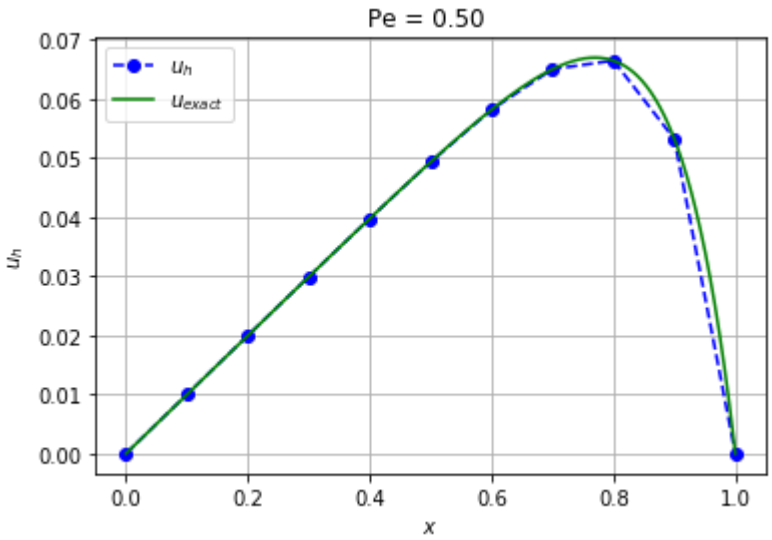
```
Pe = 0.05
Effective Pe =  0.04995837495788001
```



In [29]:

```
artificial_diffusion(epsilon=1,b=10,N=10)
```

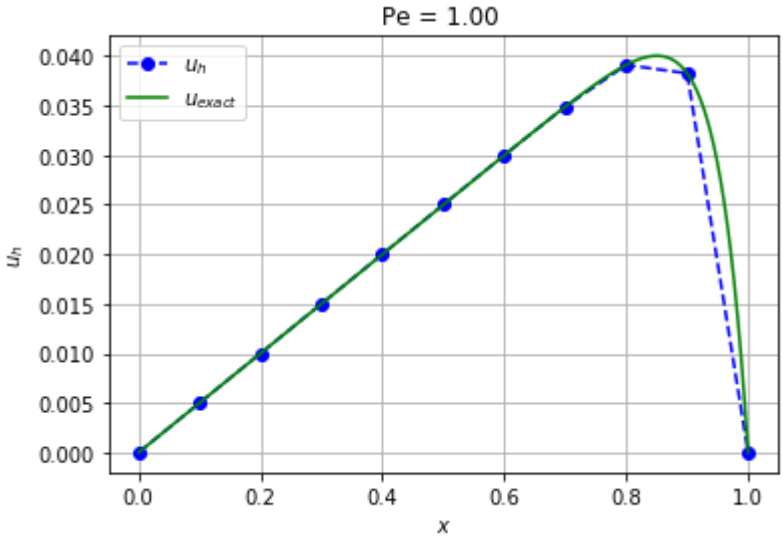
```
Pe = 0.5
Effective Pe =  0.46211715726000974
```



In [30]:

```
artificial_diffusion(epsilon=1,b=20,N=10)
```

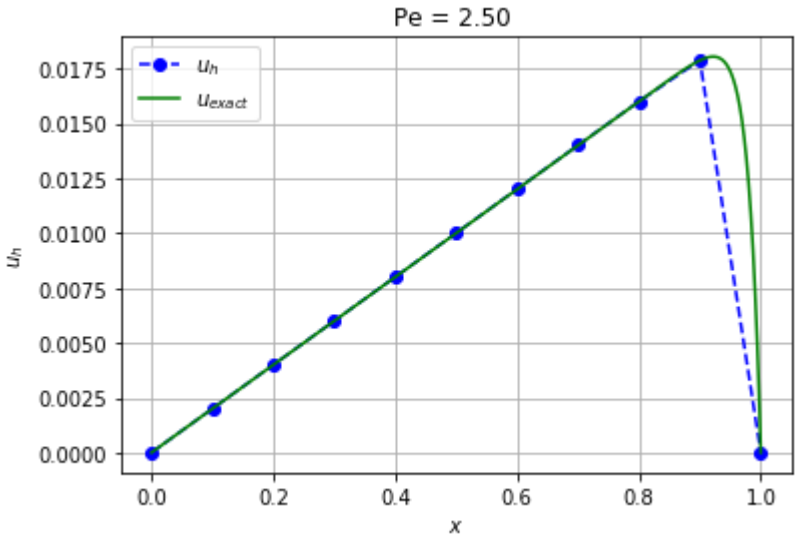
```
Pe = 1.0
Effective Pe =  0.761594155955765
```



In [31]:

```
artificial_diffusion(epsilon=1,b=50,N=10)
```

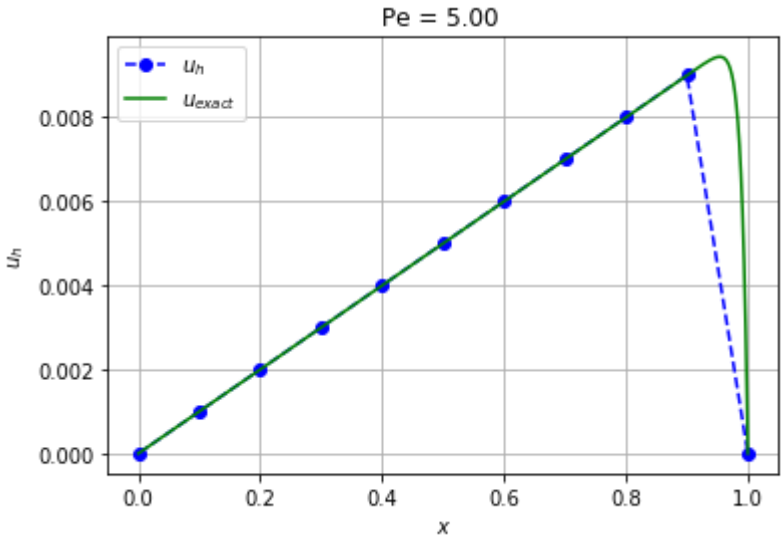
```
Pe = 2.5
Effective Pe =  0.9866142981514302
```



In [32]:

```
artificial_diffusion(epsilon=1,b=100,N=10)
```

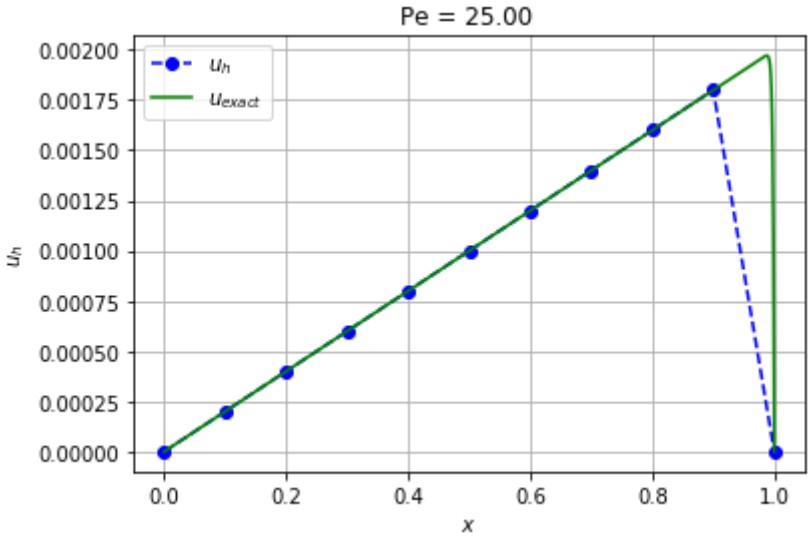
```
Pe = 5.0
Effective Pe =  0.9999092042625951
```



In [33]:

```
artificial_diffusion(epsilon=1,b=500,N=10)
```

```
Pe = 25.0
Effective Pe =  1.0
```



It can be observed that the limtations of Galerkin method have been clearly overcome by the artficial diffusion method. We do not see any oscillations in the solution at higher Peclet number Pe and we can see that the solution is stable. However, we will later see that this method does pose problems when it comes to consistency. This will be demonstrated later in Problem 2.

For now, we shall discuss another method of stabilizing the numerical solution.

3. SUPG

In this method, the modified test function $v + \delta b v'$ is used. Where δ is a mesh dependent stabilization parameter.

Multiplying by test function $v + \delta b v'$ on both sides of the convection-diffusion equation

$$\int_\Omega (-\epsilon u'' + bu')(v + \delta b v')dx = \int_\Omega (v + \delta b v')dx$$

Integrating by parts over the domain Ω

$$\epsilon \int_\Omega u'v'dx + b\int_\Omega u'vdx + \delta b^2 \int_\Omega u'v'dx = \int_\Omega (v + \delta b v')dx$$

Writing the problem in compact form:

$$u \in V: \qquad a(u,v) = l(v) \qquad \forall v \in V$$
$$a(u,v) := \epsilon \int_\Omega u'v'dx + b\int_\Omega u'vdx + \delta b^2 \int_\Omega u'v'dx$$
$$l(v) := \int_\Omega (v + \delta b v')dx$$

δ is a mesh dependent stabilization parameter. Here we choose $\delta = \frac{h}{2b}\left(\coth Pe - \frac{1}{Pe}\right)$ which gives perfect stabilization.

Let us write a function `supg` to solve the problem by adding SUPG and plots the numerical approximation against the exact solution.

In [34]:

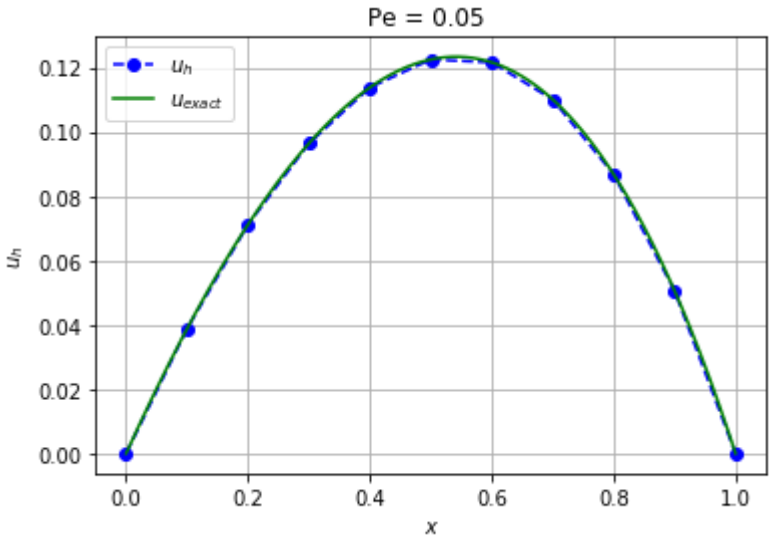
```
def supg(epsilon, b, N):
    """Solves the 1-d problem by SUPG

    Key Arguments:

    epsilon - diffusion coefficient

    b - convection coefficient

    N - mesh size

    """
    e = epsilon

    h = 1/N

    pe = peclet_number(b,N,e)
    print('Pe =',pe)

    delta = (h/(2*b))*( ((fe.exp(2*pe)+1)/(fe.exp(2*pe)-1)) - 1/pe)  # For perfect stab
ilization

    L = f*v*dx + delta*b*v.dx(0)*dx
    a = ( e * dot(grad(v),grad(u)) ) * dx + ( b * u.dx(0) * v * dx) + delta * b * b * d
ot(grad(v),grad(u)) * dx

    fe.solve(a == L, uh, bcs=bc)

    myplot(uh, b, e, pe)

    return
```

Now we shall solve the problem by SUPG method for different values of diffusion coefficient ϵ and convection coeffient b and compare the analytical and exact solution for the different values of Peclet number Pe.

In [35]:

```
supg(epsilon=1,b=1,N=10)
```

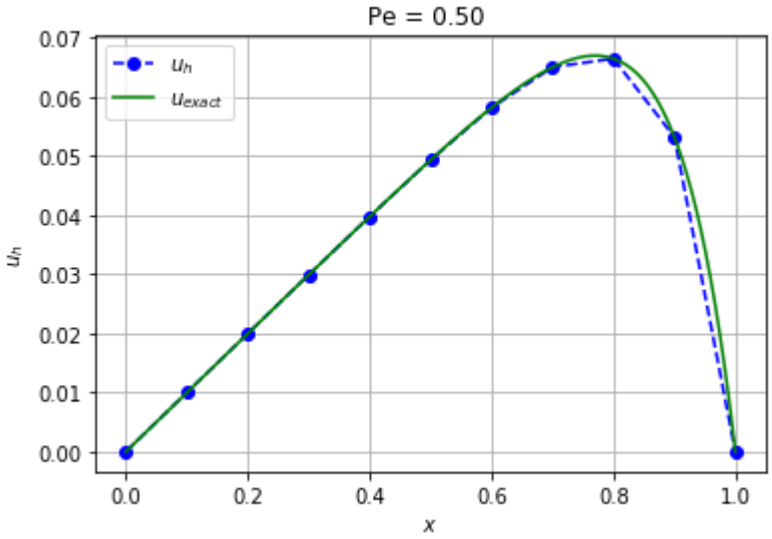
```
Pe = 0.05
```



In [36]:

```
supg(epsilon=1,b=10,N=10)
```

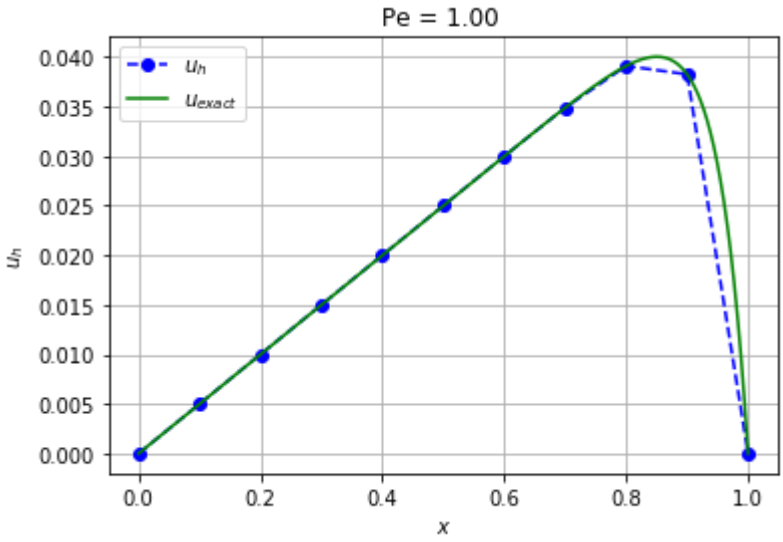
```
Pe = 0.5
```



In [37]:

```
supg(epsilon=1,b=20,N=10)
```

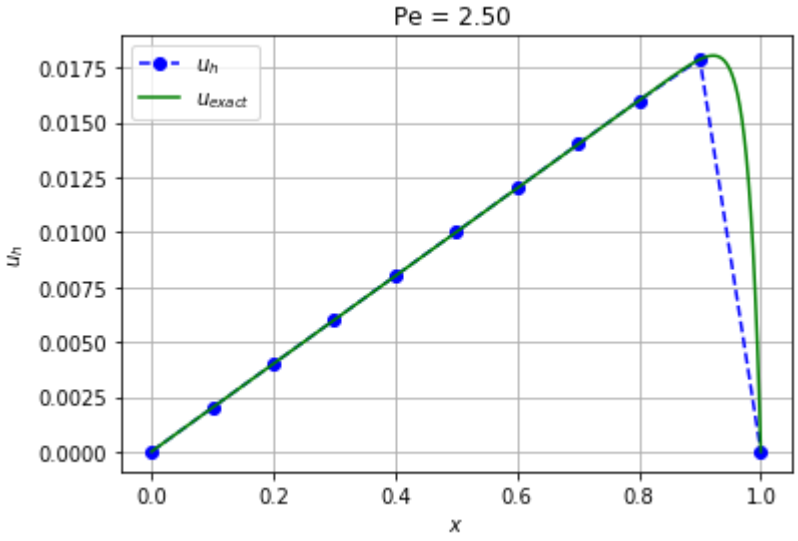
```
Pe = 1.0
```



In [38]:

```
supg(1,50,10)
```

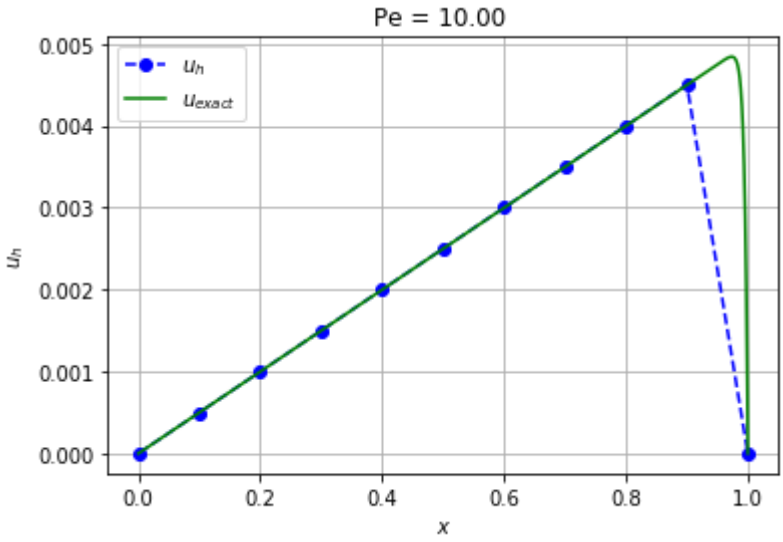
```
Pe = 2.5
```



In [39]:

```
supg(epsilon=1,b=200,N=10)
```

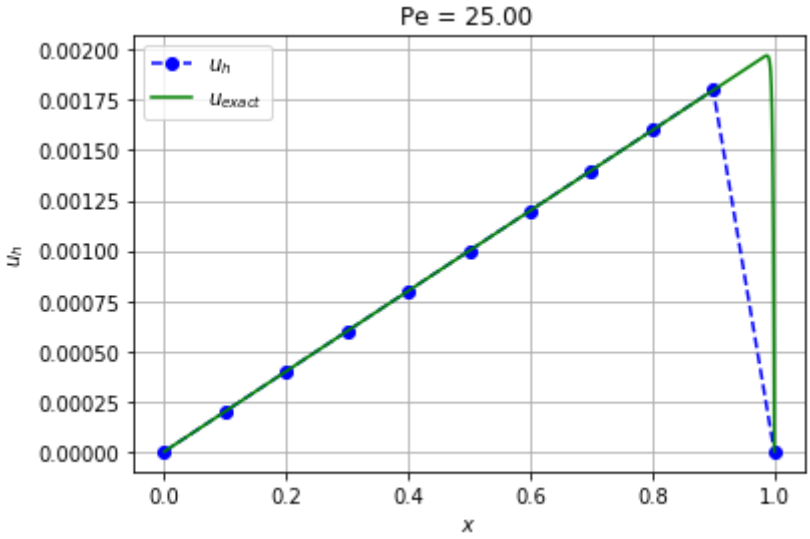
```
Pe = 10.0
```



In [40]:

```
supg(epsilon=1,b=500,N=10)
```

```
Pe = 25.0
```



It can be seen that SUPG method produces stable solutions even at higher Peclet numbers Pe. SUPG method is better than artificial diffusion method as SUPG is a strongly consistent method while artificial diffusion method is inconsistent. This is demonstrated in Problem 2 below.

Problem 2:

Consider the convection-diffusion equation :

$$-\epsilon u'' + bu' = \sin(\pi x), \quad \text{for } x \in [0,1]$$
$$u(0) = 0, \; u(1) = 1$$

where, ϵ and b are diffusion and convection coefficients respectively.

Note that the problem 2 is very similar to problem 1 with the source term being function of x (sin πx) in problem 2 while the source term was a constant in Problem 1. The boundary conditions are similar in both problems i.e Dirichlet boundary conditions in both problems. However, note that the values at the boundary are different for the problems.

1. Artificial Diffusion

Now, the variational form becomes

$$(\epsilon + \epsilon_h) \int_\Omega u'v' \, dx \; + b \int_\Omega u'v \, dx = \int_\Omega \sin(\pi x) \, v \, dx$$

where, ϵ_h is the artificial diffusion coefficient.

Now let us write the problem in compact form as follows

$$u \in V : \qquad a(u,v) = l(v) \qquad \forall v \in V$$
$$a(u,v) := (\epsilon + \epsilon_h) \int_\Omega u'v' \, dx \; + b \int_\Omega u'v \, dx$$
$$l(v) := \int_\Omega \sin(\pi x) \, v \, dx$$

Let us write a function `artificial_diffusion_sin` which solves the above problem by artfificial diffuson taking diffusion coefficient ϵ, convection coefficient b and grid size N as arguments and plots the numerical approximation against the exact solution.

In [41]:

```
def artificial_diffusion_sin(epsilon, b , N):
    """Solves the 1-d problem by Artficial Diffusion

    Key Arguments:

    epsilon - diffusion coefficient

    b - convection coefficient

    N - mesh size

    """
    e= epsilon

    mesh = fe.UnitIntervalMesh(N)
    h = 1/N
    pe = b*h/(2*e) # Peclet Number

    gamma = ( ((fe.exp(2*pe)+1)/(fe.exp(2*pe)-1)) - 1/pe)  # For perfect stabilization
    eh = gamma/2*h*b # artificial diffusion coefficient
    pe_eff = b*h/(2*(e+eh)) # Effective Peclet number

    V = fe.FunctionSpace(mesh, 'CG', 1)
    u = fe.TrialFunction(V)
    v = fe.TestFunction(V)

    x = fe.SpatialCoordinate(mesh)[0]
    pi = 3.14159
    f = fe.Function(V)
    f = fe.sin(pi*x)


    boundary_ids1 = {1} # boundaries x=0
    boundary_ids2 = {2} # boundaries x=1

    bc1 = fe.DirichletBC(V,0,boundary_ids1) # Boundary conditions for x=0, u=0
    bc2 = fe.DirichletBC(V,1,boundary_ids2) # Boundary conditions for x=1, u=1

    dx, dot, grad = fe.dx, fe.dot, fe.grad
    uh = fe.Function(V)
    L = f*v*dx
    a = ( (e+eh) * dot(grad(v),grad(u)) ) * dx + b *u.dx(0)* v * dx

    fe.solve(a == L, uh, bcs=[bc1,bc2])

    print('Pe =',pe)
    print('Effective Pe = ',pe_eff)
    print('Artficial Diffusion =',eh)

    x = np.arange(0,1.0001,0.1)

    plt.plot(x,uh(x),'--ob', label = '$u_h$')
    x = np.arange(0,1.0001,0.0001)
    aux = pi*(b**2+e**2*pi**2);
    d = np.exp(b/e);
    c1 = (-aux+b*(d+1))/(aux*(d-1));
    c2 = (aux-2*b)/(aux*(d-1));
    u_exact = c1 + c2*np.exp(b*x/e) + e*pi*(np.sin(pi*x)-b*np.cos(pi*x)/(e*pi))/aux; #
 From reference [2]
```

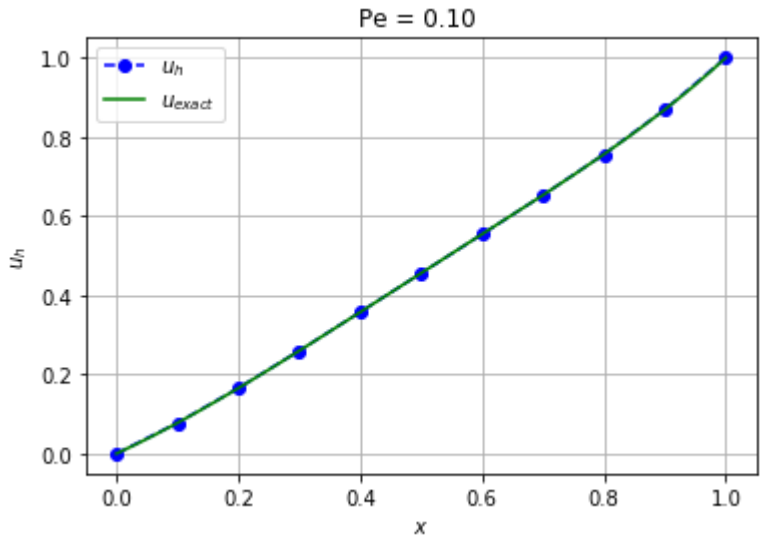
```
    plt.plot(x,u_exact,'-g', label = '$u_{exact}$')
    plt.title('Pe = %.2f '%pe)
    plt.xlabel('$x$')
    plt.ylabel('$u_h$')
    plt.legend(loc = 'upper left')
    plt.grid(True)

    return
```

Now we shall solve the above problem by artificial diffusion method for different values of diffusion coefficient ϵ and convection coeffient b and compare the analytical and exact solution for the different values of Peclet number Pe.

In [42]:

```
artificial_diffusion_sin(epsilon=0.5,b=1,N=10)
```

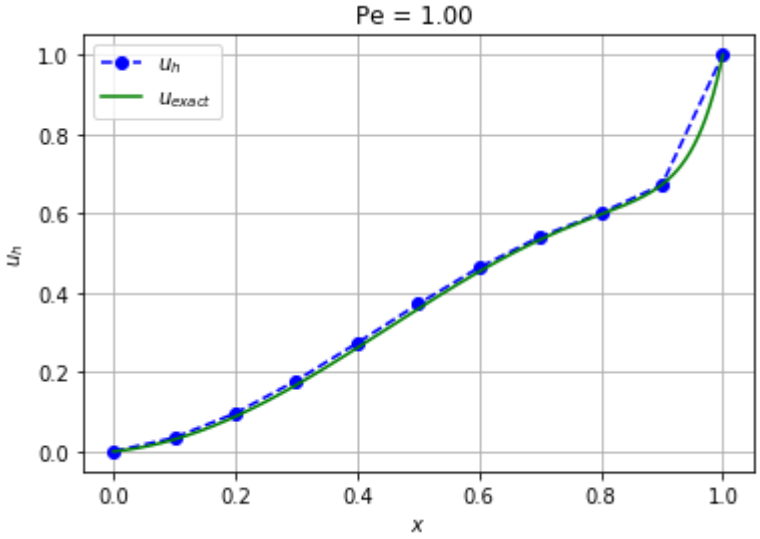
```
Pe = 0.1
Effective Pe =  0.09966799462495583
Artficial Diffusion = 0.001665556612699426
```



In [43]:

```
artificial_diffusion_sin(epsilon=0.05,b=1,N=10)
```

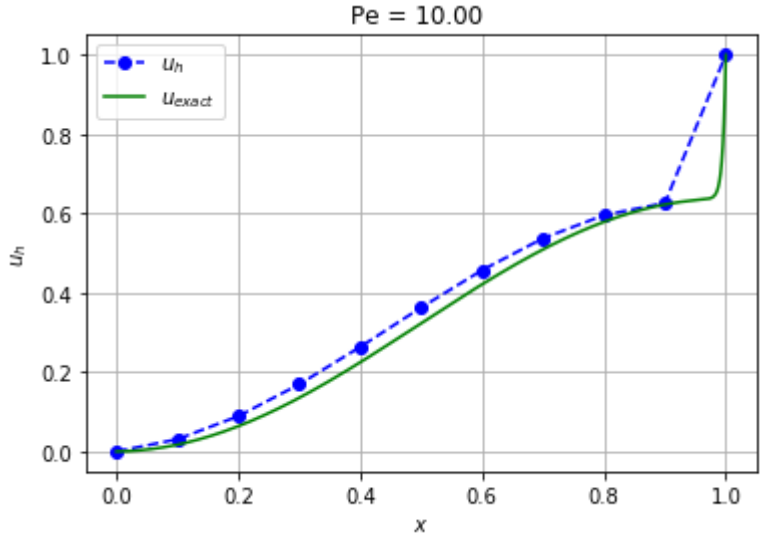
```
Pe = 1.0
Effective Pe =  0.761594155955765
Artficial Diffusion = 0.015651764274966562
```



In [44]:

```
artificial_diffusion_sin(epsilon=0.005,b=1,N=10)
```

```
Pe = 10.0
Effective Pe =  0.999999995877693
Artficial Diffusion = 0.04500000020611536
```



It can be seen that artificial diffusion method does not give accurate results for higher Peclet numbers. This is mainly because we are not solving the strong form exactly. In other words, the variational form is not consistent with the strong form. We shall see that SUPG is a strongly consistent method.

2. SUPG

Multiplying by test function $v + \delta\, b\, v'$ on both sides of the convection-diffusion equation

$$\int_\Omega (-\epsilon u'' + bu')(v + \delta b v')dx = \int_\Omega \sin(\pi x)\,(v + \delta b v')dx$$

Integrating over domain Ω

$$\epsilon \int_\Omega u'v'\,dx + b\int_\Omega u'v dx + \delta b^2 \int_\Omega u'v'\,dx = \int_\Omega \sin(\pi x)\,v\,dx + \delta b \int_\Omega \sin(\pi x)\,v'\,dx$$

Now let us write the problem in compact form as follows

$$u \in V: \qquad a(u,v) = l(v) \quad \forall v \in V$$
$$a(u,v) := \epsilon \int_\Omega u'v'\,dx + b\int_\Omega u'v dx + \delta b^2 \int_\Omega u'v'\,dx$$
$$l(v) := \int_\Omega \sin(\pi x)\, v\, dx + \delta b \int_\Omega \sin(\pi x)\, v'\, dx$$

Let us write a function `supg_sin` which solves the above problem by SUPG taking diffusion coefficient ϵ, convection coefficient b and grid size N as arguments and plots the numerical approximation against the exact solution.

In [45]:

```
def supg_sin(epsilon, b, N):
    """Solves the 1-d problem by SUPG

    Key Arguments:

    epsilon - diffusion coefficient

    b - convection coefficient

    N - mesh size

    """
    e = epsilon

    mesh = fe.UnitIntervalMesh(N)

    V = fe.FunctionSpace(mesh, 'CG', 1)
    u = fe.TrialFunction(V)
    v = fe.TestFunction(V)

    x = fe.SpatialCoordinate(mesh)[0]

    pi = 3.14159
    f = fe.Function(V)
    f = fe.sin(pi*x)

    boundary_ids1 = {1} # boundaries x=0
    boundary_ids2 = {2} # boundaries x=1

    bc1 = fe.DirichletBC(V,0,boundary_ids1) # Boundary conditions for x=0, u=0
    bc2 = fe.DirichletBC(V,1,boundary_ids2) # Boundary conditions for x=1, u=1

    h = 1/N
    pe = b/(2*e*N)
    dell = (h/(2*b))*( ((fe.exp(2*pe)+1)/(fe.exp(2*pe)-1)) - 1/pe)

    dx, dot, grad = fe.dx, fe.dot, fe.grad
    uh = fe.Function(V)
    L = f*v*dx + dell*b*f*v.dx(0)*dx
    a = ( e * dot(grad(v),grad(u)) ) * dx + ( b * u.dx(0) * v * dx) + dell * b * b * do
t(grad(v),grad(u)) * dx

    fe.solve(a == L, uh, bcs=[bc1,bc2])

    print('Pe =',pe)

    x = np.arange(0,1.0001,0.1)
    plt.plot(x,uh(x),'--ob', label = '$u_h$')
    x = np.arange(0,1.0001,0.0001)
    aux = pi*(b**2+e**2*pi**2);
    d = np.exp(b/e);
    c1 = (-aux+b*(d+1))/(aux*(d-1));
    c2 = (aux-2*b)/(aux*(d-1));
    u_exact = c1 + c2*np.exp(b*x/e) + e*pi*(np.sin(pi*x)-b*np.cos(pi*x)/(e*pi))/aux;  #
from refrence [2]

    plt.plot(x,u_exact,'-g', label = '$u_{exact}$')
    plt.title('Pe = %.2f '%pe)
    plt.xlabel('$x$')
```

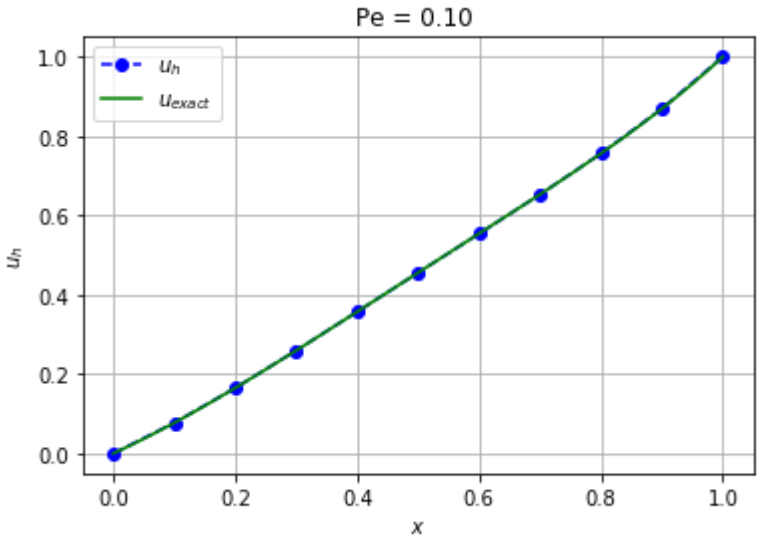
```
    plt.ylabel('$u_h$')
    plt.legend(loc = 'upper left')
    plt.grid(True)

    return
```

Now we shall solve the above problem by SUPG method for different values of diffusion coefficient ϵ and convection coeffient b and compare the analytical and exact solution for the different values of Peclet number Pe.

In [46]:

```
supg_sin(epsilon=0.5,b=1,N=10)
```

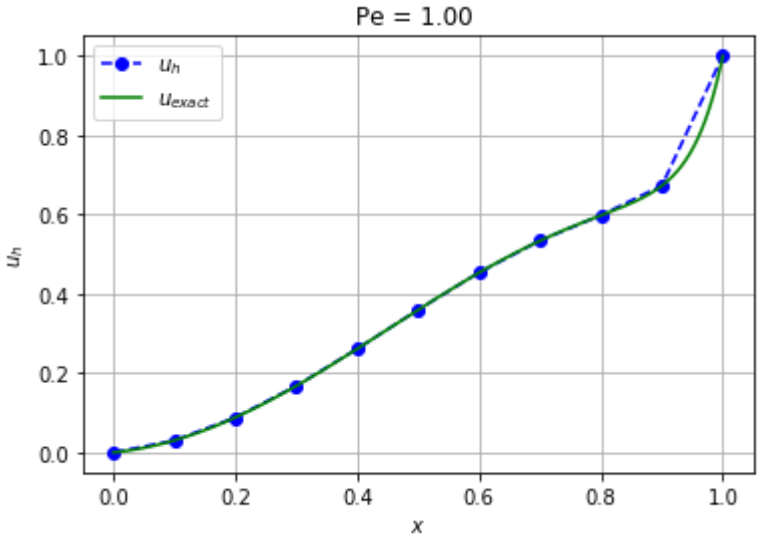
```
Pe = 0.1
```



In [47]:

```
supg_sin(epsilon=0.05,b=1,N=10)
```

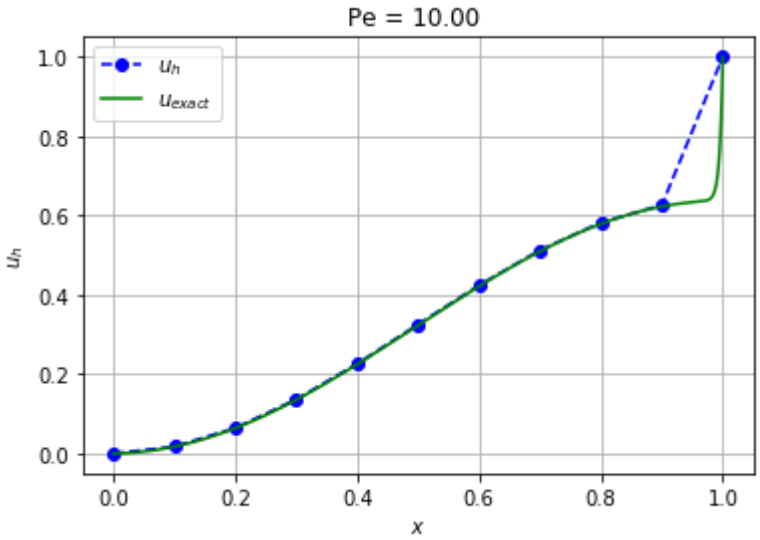
```
Pe = 1.0
```



In [48]:

```
supg_sin(epsilon=0.005,b=1,N=10)
```

```
Pe = 10.0
```



It can be seen the SUPG produces consistent solution even for higher Peclet numbers Pe.

Conclusion

-> It can be seen that Galerkin method is not suited for convection dominated problems i.e $Pe > 1$.

-> Artificial diffusion method produces stable solution but lacks consistency.

-> SUPG is a strongly consistent method.

Let us now consider a 2d problem to see how the three methods discussed above produces solutions in 2d.

Problem 3: 2d Problem

Consider the 2d convection-diffusion equation with dirichlet boundary condition :

$$\begin{aligned} -\epsilon\nabla^2 u + b.\nabla u = 0, \quad \text{in} \quad \Omega = [0,1]\times[0,1] \\ u = 1 \quad \forall x = 0 \\ u = 0 \quad \forall y = 0 \end{aligned}$$

where, ϵ and b are diffusion and convection coefficients respectively. We seek to find the solution $u \in V$, where V is some finite element space satisfying the boundary condition.

1. Galerkin

We write the above problem in variational form by multiplying by test function $v \in V$ and integrating by parts over domain Ω

$$\epsilon \iint_\Omega (\nabla u.\nabla v)\, dx\, dy + \iint_\Omega (b.\nabla u)\ v\, dx\, dy = 0$$

Now let us write the problem in compact form as follows

$$u \in V: \quad a(u,v) = l(v) \qquad \forall v \in V$$
$$a(u,v) := \epsilon \iint_\Omega (\nabla u . \nabla v)\, dx\, dy + \iint_\Omega (b . \nabla u)\, v\, dx\, dy$$
$$l(v) := 0$$

Let us write a function `galerkin2d` to solve the above problem by galerkin. Let the convection coeffient b be $(0.866, 0.5)$ i.e 0.866 in x-direction and 0.5 in in y-direction respectively. The diffusion coefficient ϵ and grid size N will be passed as arguments to the function.

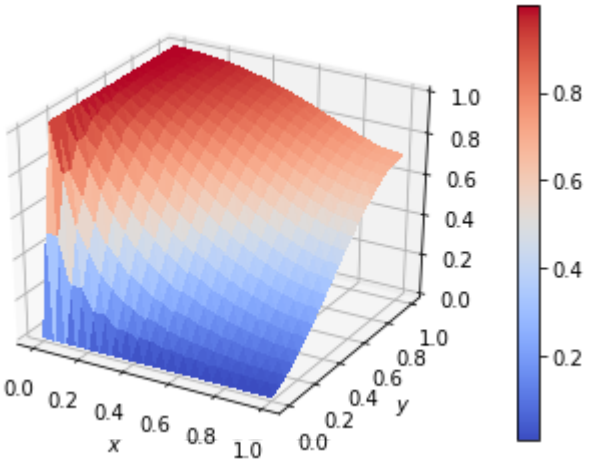
In [49]:

```
def galerkin2d(epsilon,N):
    """Solves the 2-d problem by Galerin

    Key Arguments:

    epsilon - diffusion coefficient

    N - mesh size

    """

    e = epsilon

    mesh = fe.UnitSquareMesh(N,N)

    V = fe.FunctionSpace(mesh, 'CG', 1)
    u = fe.TrialFunction(V)
    v = fe.TestFunction(V)
    x,y = fe.SpatialCoordinate(mesh)

    f = fe.Constant(0)

    b = 0.86*fe.unit_vectors(2)[0] + 0.5*fe.unit_vectors(2)[1] #convection constant

    boundary_ids1 = {1} # boundaries x=0 and x=1
    boundary_ids2 = {3} # boundaries y=0 and y=1

    bc1 = fe.DirichletBC(V,1,boundary_ids1) # Boundary conditions for x=0
    bc2 = fe.DirichletBC(V,0,boundary_ids2) # Boundary conditions for y=0

    uh = fe.Function(V)
    L = f* v *dx
    a =  e * dot( grad(v), grad(u))  * dx + dot(b, grad(u)) * v * dx

    fe.solve(a == L, uh, bcs=[bc1,bc2])
    fe.plot(uh, plot3d = 'TRUE')
    plt.xlabel('$x$')
    plt.ylabel('$y$')

    return
```

Now we shall solve the problem for different values of diffusion coefficient ϵ and convection coeffient b and plot the numerical approximation for different Peclet numbers Pe.

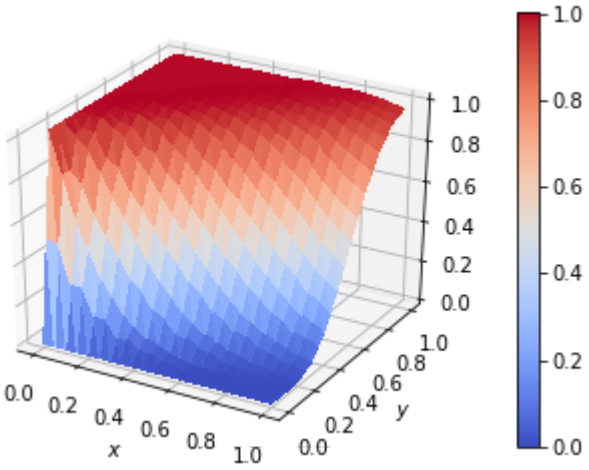
In [50]:

```
galerkin2d(epsilon=0.1,N=10)
```



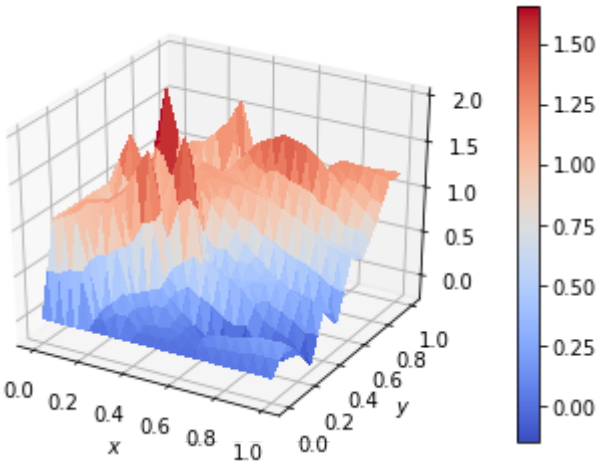
In [51]:

```
galerkin2d(epsilon=0.01,N=10)
```



In [52]:

```
galerkin2d(epsilon=0.001,N=10)
```



As expected, the galerkin method produces oscillations for higher Peclet numbers Pe.

2. Artficial diffusion

In this method, an artificial diffusion is added such that it reduces the mesh Peclet number to an effective Peclet number of almost unity.

Now, the variational form becomes

$$(\epsilon + \epsilon_h) \iint_\Omega (\nabla u . \nabla v) dx\, dy + \iint_\Omega (b . \nabla u)\ v\, dx\, dy = 0$$

where, ϵ_h is the artificial diffusion coefficient.

Now let us write the problem in compact form as follows

$$u \in V : \quad a(u,v) = l(v) \qquad \forall v \in V$$
$$a(u,v) := (\epsilon + \epsilon_h) \iint_\Omega (\nabla u . \nabla v) dx\, dy + \iint_\Omega (b . \nabla u)\ v\, dx\, dy$$
$$l(v) := 0$$

The value of ϵ_h is chosen such that it reduces the Peclet number. Choose $\epsilon_h = \frac{\gamma}{2} hb$.

If $\gamma \geq 1$, the effective Peclet number is sufficiently small and hence diffusion is minimized.

For perfect stabilization use $\gamma = \coth(Pe) - 1/Pe$

Let us write a function `artificial_diffusion2d` to solve the above 2d problem by artificial diffusion. Let the convection coeffient b be $(0.866, 0.5)$ i.e 0.866 in x-direction and 0.5 in in y-direction respectively. The diffusion coefficient ϵ and grid size N will be passed as arguments to the function.

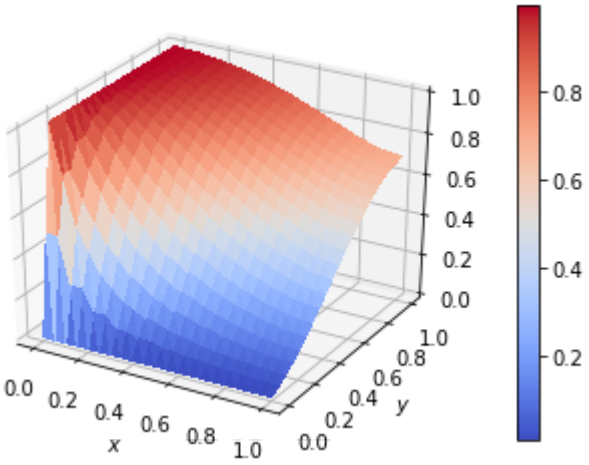
In [53]:

```
def artificial_diffusion2d(epsilon,N):
    """Solves the 2-d problem by Artificial diffusion

    Key Arguments:

    epsilon - diffusion coefficient

    N - mesh size

    """

    e = epsilon
    h = 1/N

    mesh = fe.UnitSquareMesh(N,N)

    V = fe.FunctionSpace(mesh, 'CG', 1)
    u = fe.TrialFunction(V)
    v = fe.TestFunction(V)
    x,y = fe.SpatialCoordinate(mesh)

    f = fe.Constant(0)

    b = 0.86*fe.unit_vectors(2)[0] + 0.5*fe.unit_vectors(2)[1] #convection constant

    boundary_ids1 = {1} # boundaries x=0 and x=1
    boundary_ids2 = {3} # boundaries y=0 and y=1

    bc1 = fe.DirichletBC(V,1,boundary_ids1) # Boundary conditions for x=0
    bc2 = fe.DirichletBC(V,0,boundary_ids2) # Boundary conditions for y=0

    pe = fe.sqrt(fe.dot(b,b))*h/(2*fe.sqrt(fe.dot(e,e))) # Peclet Number
    gamma = ( ((fe.exp(2*pe)+1)/(fe.exp(2*pe)-1)) - 1/pe)  # For perfect stabilization
    eh = gamma/2*h*fe.sqrt(fe.dot(b,b)) # artificial diffusion coefficient

    uh = fe.Function(V)
    L = f * v * dx
    a =   (e+eh) * dot( grad(u), grad(v))  * dx  + dot(b, grad(u)) * v * dx

    fe.solve(a == L, uh, bcs=[bc1,bc2])

    fe.plot(uh, plot3d = 'TRUE')
    plt.xlabel('$x$')
    plt.ylabel('$y$')

    return
```

Now we shall solve the above problem by artificial diffusion method for different values of diffusion coefficient ϵ and plot the numerical approximation for different Peclet numbers Pe.

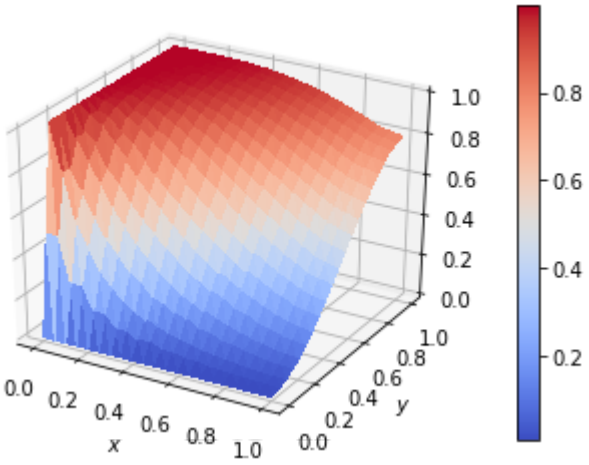
In [54]:

```
artificial_diffusion2d(epsilon=0.1,N=10)
```



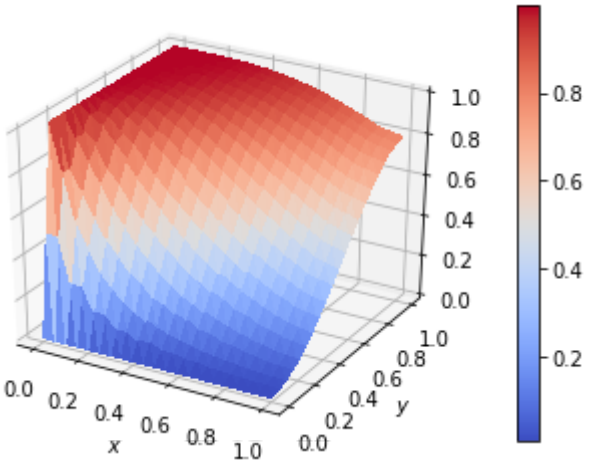
In [55]:

```
artificial_diffusion2d(epsilon=0.01,N=10)
```



In [56]:

```
artificial_diffusion2d(epsilon=0.001,N=10)
```



We see that artificial diffusion method produces stable solution. However we do notice higher diffusion in this method due to the additional artificial diffusion added to the variational form.

3. SUPG

In this method, the modified test function $v + \delta b \nabla v$ is used. Where δ is a mesh dependent stabilization parameter.

Multiplying by test function $v + \delta b \nabla v$ on both sides of the convection-diffusion equation

$$\iint_\Omega (-\epsilon \nabla^2 u + b.\,\nabla u)(v + \delta\, b.\,\nabla v) dx\, dy = 0$$

Integrating by parts over domain Ω

$$\epsilon \iint_\Omega (\nabla u.\,\nabla v)\, dx\, dy + \iint_\Omega (b.\,\nabla u)\, v\, dx\, dy + \delta \iint_\Omega (b.\,\nabla u)(b.\,\nabla v)\, dx\, dy = 0$$

Now let us write the problem in compact form as follows

$$u \in V: \qquad a(u,v) = l(v) \qquad \forall v \in V$$

$$a(u,v) := \epsilon \iint_\Omega (\nabla u.\,\nabla v)\, dx\, dy + \iint_\Omega (b.\,\nabla u)\, v\, dx\, dy + \delta \iint_\Omega (b.\,\nabla u)(b.\,\nabla v)\, dx\, dy$$

$$l(v) := 0$$

δ is a mesh dependent stabilization parameter. Choose $\delta = \frac{h}{2b}\left(\coth Pe - \frac{1}{Pe}\right)$

Let us write a function `supg2d` to solve the above 2d problem by SUPG. Let the convection coefficient b be $(0.866, 0.5)$ i.e 0.866 in x-direction and 0.5 in in y-direction respectively. The diffusion coefficient ϵ and grid size N will be passed as arguments to the function.

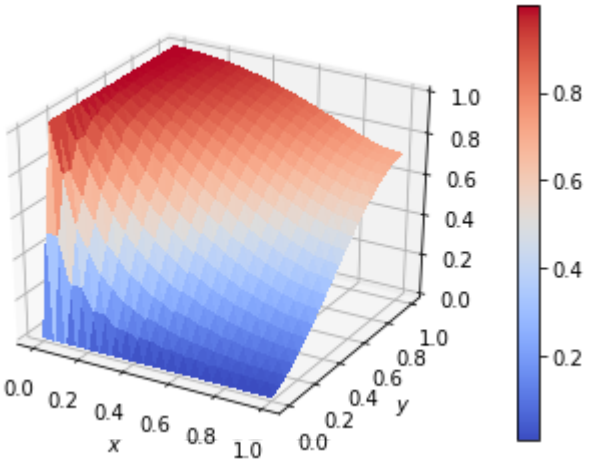
In [57]:

```
def supg2d(epsilon,N):
    """Solves the 2-d problem by SUPG

    Key Arguments:

    epsilon - diffusion coefficient

    N - mesh size

    """

    e = epsilon
    h = 1/N

    mesh = fe.UnitSquareMesh(N,N)

    V = fe.FunctionSpace(mesh, 'CG', 1)
    u = fe.TrialFunction(V)
    v = fe.TestFunction(V)
    x,y = fe.SpatialCoordinate(mesh)

    f = fe.Constant(0)

    b = 0.86*fe.unit_vectors(2)[0] + 0.5*fe.unit_vectors(2)[1] #convection constant

    boundary_ids1 = {1} # boundaries x=0 and x=1
    boundary_ids2 = {3} # boundaries y=0 and y=1

    bc1 = fe.DirichletBC(V,1,boundary_ids1) # Boundary conditions for x=0
    bc2 = fe.DirichletBC(V,0,boundary_ids2) # Boundary conditions for y=0

    pe = fe.sqrt(fe.dot(b,b))*h/2/e # Peclet Number
    gamma = ( ((fe.exp(2*pe)+1)/(fe.exp(2*pe)-1)) - 1/pe)  # For perfect stabilization
    delta = h/(2*fe.sqrt(fe.dot(b,b)))*( ((fe.exp(2*pe)+1)/(fe.exp(2*pe)-1)) - 1/pe)

    uh = fe.Function(V)
    L = f*v*dx + delta*dot(b,grad(v))*f*dx
    a =  e * dot( grad(v), grad(u)) * dx + dot(b,grad(u)) * v * dx + delta * dot(b,grad
(u))*dot(b,grad(v)) * dx

    fe.solve(a == L, uh, bcs=[bc1,bc2])

    fe.plot(uh, plot3d = 'TRUE')
    plt.xlabel('$x$')
    plt.ylabel('$y$')

    return
```

Now we shall solve the above problem by SUPG method for different values of diffusion coefficient ϵ and convection coeffient b and compare the analytical and exact solution for the different values of Peclet number Pe.

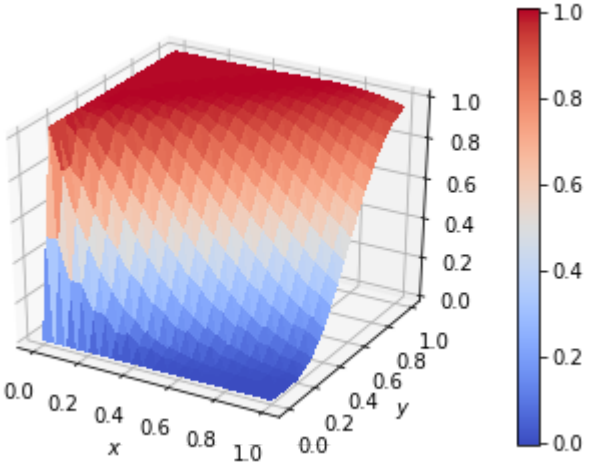
In [58]:

```
supg2d(epsilon=0.1,N=10)
```



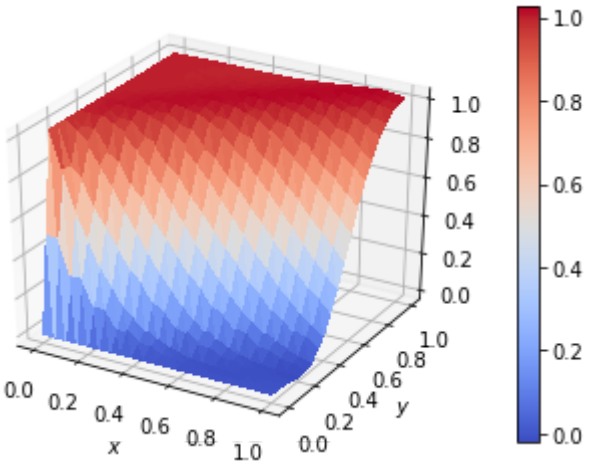
In [59]:

```
supg2d(epsilon=0.01,N=10)
```



In [60]:

```
supg2d(epsilon=0.001,N=10)
```



We see that SUPG method produces stable solutions. We can also observe that SUPG method produces solution with lesser diffusion when compared to artificial difusion method.

Conclusion

It is seen that the galerkin method is unstable and produces oscillatory results for convection dominated problems. Artificial diffusion method is more stable but produces excessive diffusion in the results. SUPG method produces conistent and stable results.

References

1. Finite Element Methods fo Fluif flow Problems - Jean Donea and Antonio Huerta
2. http://ww2.lacan.upc.edu/huerta/exercises/SteadyTransport/SteadyTransport.htm (http://ww2.lacan.upc.edu/huerta/exercises/SteadyTransport/SteadyTransport.htm)