Problem 1 - St. Venant-Kirchoff Hyperelasticity (manufactured solution)

In [1]:
```
from dolfin import *
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
N = 8
mesh = UnitCubeMesh(N,N,N)
X = SpatialCoordinate(mesh)
k = 1
dx = dx(metadata ={'quadrature_degree':2*k})
V = VectorFunctionSpace (mesh , "CG",k)
u = Function (V)
I = Identity(len (u))
def problem(u):
    I = Identity(len (u))
    F = I + grad(u) # deformation gradient
    C = F.T*F # Cauchy-Green tensor
    E = 0.5*(C-I) # Green-Lagrange Strain tensor
    K = Constant(1.0e1)
    mu = Constant(1.0e1)
    S = K*tr(E)*I + 2.0*mu*(E - tr(E)*I/3.0) # 2nd PK Stress tensor
    psi = 0.5*inner(E,S)
    return F,S,psi
err_dict = dict()
```

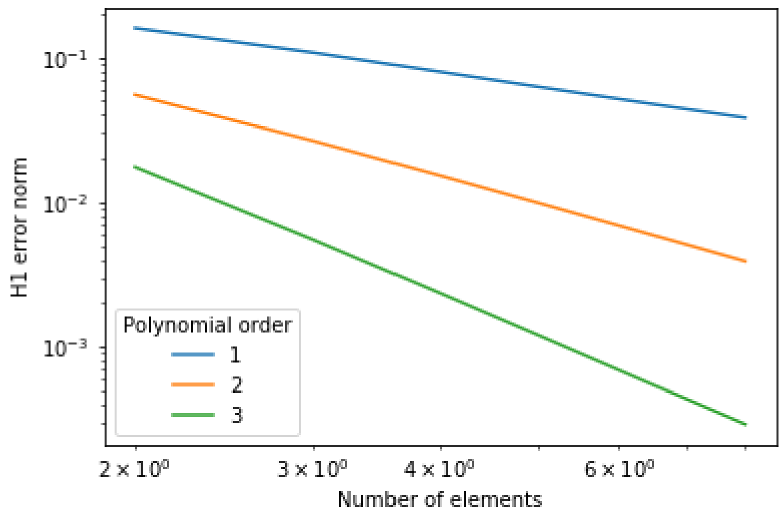
In [3]:
```
u_ex = as_vector(3*[0.1*sin(pi*X[0]) * sin(pi*X[1])*sin(pi*X[2]) ,])
H1err = lambda u: sqrt(assemble(((u-u_ex)**2 + (div(u)-div(u_ex))**2)*dx
))
```

In [4]:
```
kvec = [1,2,3]
Nvec = [x for x in range(2,9)]
for k in kvec: # order of polynomials
    err_dict[k] = []
    print(f"Errors for polynomial order {k}:")
    for N in Nvec:
        mesh = UnitCubeMesh(N,N,N)
        X = SpatialCoordinate(mesh)
        u_ex = as_vector(3*[0.1*sin(pi*X[0]) * sin(pi*X[1])*sin(pi*X[2])
,])

        dx = dx(metadata ={'quadrature_degree':2*k})
        V = VectorFunctionSpace (mesh , "CG",k)
        u = Function (V)
        I = Identity(len (u))
        F_ex, S_ex, psi_ex = problem(u_ex)
        P_ex = F_ex * S_ex
        f0 = -div(P_ex)

        F,S,psi = problem(u)
        v = TestFunction(V)
        R = derivative(psi,u,v)*dx - inner(f0,v)*dx
        J = derivative(R,u)
        bc = DirichletBC(V, Constant((0 ,0 ,0)) ,"on_boundary")
        solve(R==0,u ,[ bc ,] , J=J )

        #err = sqrt(assemble(((u-u_ex)**2 + (div(u)-div(u_ex))**2)*dx))
        err = H1err(u)
        print(f"N = {N}: {err}")
        err_dict[k].append(err)
```

```
Errors for polynomial order 1:
N = 2: 0.16002836571983092
N = 3: 0.10823162759846593
N = 4: 0.07956894407298544
N = 5: 0.06269450715439057
N = 6: 0.05173568767763485
N = 7: 0.04405970786877834
N = 8: 0.03838219899645828
Errors for polynomial order 2:
N = 2: 0.055264797053366385
N = 3: 0.02652352906257629
N = 4: 0.015272356632708284
N = 5: 0.009881982580419169
N = 6: 0.006899259320880784
N = 7: 0.005083311974443078
N = 8: 0.0038982721468753235
Errors for polynomial order 3:
N = 2: 0.0175292861835237
N = 3: 0.0054932682461913095
N = 4: 0.0023347735539772717
N = 5: 0.0011943853784155201
N = 6: 0.0006896614308533662
N = 7: 0.0004334982984876113
N = 8: 0.0002899940169290507
```

In [5]:
```
for k in kvec:
    plt.loglog(Nvec, err_dict[k],label=k)
    slope, _ = np.polyfit(np.log(Nvec), np.log(err_dict[k]), 1)
    print(f"Convergence rate (fitted) of the error in the H1 norm for po
ly order (k={k}) is {-slope}")

plt.xlabel('Number of elements')
plt.ylabel('H1 error norm')
plt.legend(title="Polynomial order")
plt.show()
```

```
Convergence rate (fitted) of the error in the H1 norm for poly order (k
=1) is 1.0378470925934706
Convergence rate (fitted) of the error in the H1 norm for poly order (k
=2) is 1.9191775648499703
Convergence rate (fitted) of the error in the H1 norm for poly order (k
=3) is 2.9665925459534104
```



As expected, the H1 error converges on the order of h, h^2, and h^3 for polynomial bases orders of 1, 2, and 3, respectively.

Problem 2 - Using Gateaux Derivative to Derive Jacobian

$$d\psi(u; v) = (f_0, v)$$
$$R = 0 = d\psi(u; v) - (f_0, v)$$

Where ψ is the energy potential function

In [6]:
```
from ufl import indices
```

In [7]:

```
N = 8
mesh = UnitCubeMesh(N,N,N)
X = SpatialCoordinate(mesh)
dx = dx(metadata ={'quadrature_degree':2})
V = VectorFunctionSpace(mesh,"CG",1)

du = TrialFunction(V)
u = Function(V)
v = TestFunction(V)

def manual_problem(u):
    i,j,s = indices(3) # s is a dummy index

    K = Constant(1.0)
    mu = Constant(1.0)
    I = Identity(len(u))
    F = as_tensor(I[i,j] + u[i].dx(j), (i,j))#I + grad(u) # deformation
 gradient
    C = as_tensor(F[s,j]*F[s,i], (i,j))
    E = as_tensor(0.5*(C[i,j]-I[i,j]), (i,j)) # Green-Lagrange Strain te
nsor
    S = as_tensor(K*tr(E)*I[i,j] + 2.0*mu*(E[i,j] - tr(E)*I[i,j]/3.0), (
i,j)) # 2nd PK Stress tensor
    psi = 0.5*E[i,j]*S[i,j]
    return F,S,psi

_,_,psi = manual_problem(u)
u_ex = as_vector(3*[0.1*sin(pi*X[0]) * sin(pi*X[1])*sin(pi*X[2]) ,])
F_ex, S_ex, psi_ex = manual_problem(u_ex)
P_ex = F_ex * S_ex
f0 = -div(P_ex) # using manufactured solution
R = derivative(psi,u,v)*dx - inner(f0,v)*dx
bc = DirichletBC(V, Constant((0 ,0 ,0)) ,"on_boundary")

def manual_jacobian(u,V):
    du = TrialFunction(V)
    v = TestFunction(V)
    i,j,k,l,s = indices(5) # s is a dummy index
    K = Constant(1.0)
    mu = Constant(1.0)
    I = Identity(len(u))
    Ctensor = as_tensor(K*I[i,j]*I[k,l] + mu*(I[i,k]*I[j,l] + I[i,l]*I[j
,k]), (i,j,k,l)) # rank 4 isotropic tensor
    F = as_tensor(I[i,j] + u[i].dx(j), (i,j))#I + grad(u) # deformation
 gradient
    #C = F.T*F # Cauchy-Green tensor
    C = as_tensor(F[s,j]*F[s,i], (i,j))
    E = as_tensor(0.5*(C[i,j]-I[i,j]), (i,j)) # Green-Lagrange Strain te
nsor
    S = as_tensor(K*tr(E)*I[i,j] + 2.0*mu*(E[i,j] - tr(E)*I[i,j]/3.0), (
i,j)) # 2nd PK Stress tensor

    J = (1/2*(du[s].dx(i)*v[s].dx(j) + du[s].dx(j)*v[s].dx(i))*Ctensor[i
,j,k,l]*E[k,l] \
        + 1/4*(v[i].dx(j)+v[j].dx(i) + u[s].dx(i)*v[s].dx(j) + u[s].dx(j
)*v[s].dx(i))*Ctensor[i,j,k,l]*(du[k].dx(l) + du[l].dx(k) + u[s].dx(k)*d
```

```
u[s].dx(l) + u[s].dx(l)*du[s].dx(k)))*dx
    return J
J = manual_jacobian(u,V)
```

In [8]:
```
A,L = assemble_system(-J, R, bc)
ddu = Function(V)
u = Function(V)
niter = 5

for idx in range(niter):
    solve(A, ddu.vector(), L)
    u.assign(u+ddu)

    J = manual_jacobian(u,V)
    _,_,psi = manual_problem(u)
    R = derivative(psi,u,v)*dx - inner(f0,v)*dx
    A,L = assemble_system(-J, R, bc)
    print(H1err(u))
```

```
0.06175998778164446
0.038942769128707054
0.03836040397035979
0.038375167916049495
0.038381042883361395
```

H1 error converges... See attached image "manual_gateaux_derivative.jpg" for derivation

Problem 3 - Updated Lagrangian Approach

Now we simply use the 2nd (?...undeformed force in undeformed area?) PK stress tensor in the formulation of R

In [9]:
```
N = 8
mesh = UnitCubeMesh(N,N,N)
X = SpatialCoordinate(mesh)
dx = dx(mesh,metadata ={'quadrature_degree':2})
V = VectorFunctionSpace(mesh,"CG",1)

du = TrialFunction(V)
u = Function(V)
v = TestFunction(V)

F,S,psi = manual_problem(u)
u_ex = as_vector(3*[0.1*sin(pi*X[0]) * sin(pi*X[1])*sin(pi*X[2]) ,])
F_ex, S_ex, psi_ex = manual_problem(u_ex)
P_ex = F_ex * S_ex
f0 = -div(S_ex) # using manufactured solution
R = inner(S,grad(v))*dx - inner(f0,v)*dx
bc = DirichletBC(V, Constant((0 ,0 ,0)) ,"on_boundary")
```

In [10]:
```
#J = derivative(R,u,v)
J = manual_jacobian(u,V)
A,L = assemble_system(-J, R, bc)
ddu = Function(V)
u = Function(V)
niter = 5

for idx in range(niter):
    solve(A, ddu.vector(), L)
    u.assign(u+ddu)

    ALE.move(mesh,ddu)

    F,S,psi = manual_problem(u)
    R = inner(S,grad(v))*dx - inner(f0,v)*dx
    A,L = assemble_system(-J, R, bc)
    print(H1err(u))
```

```
0.06751443024668852
0.04480882596283044
0.04143984100009732
0.041092986284665345
0.04116197517643754
```