Programming Project Report

Anastasia Papadaki (22902210)
Supervisor Dr.Paolo Moretti
Friedrich Alexander University Erlangen-Nuremberg
- WW8-Institute of Materials Simulation

Abstract

In the present work, we study the implementation of a Random Fuse Model problem in a square lattice, similar to that introduced in the CoMBiF course, using the jupyter notebook ProgrammingProject.ipynb.
Our system is subject to a uniaxial load in the vertical direction, with constant voltages $V = 1$ and $V = 0$ applied to the top and bottom boundary respectively. Periodic boundary conditions are applied in the horizontal direction. This configuration mimics the geometry of an elastic/brittle material, which is subject to tension in the vertical direction.
However, because of the crack, some currents flow in the horizontal direction, mimicking load redistribution around the crack profile, in analogy with patterns of stress redistribution in fracture mechanics.

Keywords Biomaterial failure, python, Anaconda, fuse model, square lattice.

1 Introduction

It is true that materials and materials science have played an important role in people's everyday life since Stone Age. More specifically originating primarily from ceramic manufacturing, materials science is one of the oldest forms of engineering and applied science. Modern materials science evolved directly from metallurgy, which itself evolved from mining and (probably) ceramics, and earlier from the use of fire. A major breakthrough in our understanding of materials occurred in the late 19th century, when the American scientist Josiah Willard Gibbs demonstrated that thermodynamic properties related to atomic structure in different phases are related to the physical properties of a material. The interdisciplinary field of materials science covers the design and discovery of new materials, in particular solids.[1].
An important division of materials science is the biomaterials. A biomaterial is any matter, surface, or construct that interacts with biological systems. Biomaterials can be found or derived in nature and they can be also synthesised for different purposes in bioengineering and especially Tissue Engineering in Regenerative Medicine. The application is very wide, thus they can be used in heart electrical impulse generator (pace makers), prostheses, in drug delivery systems as capsules or nanoshells, or microbasket for carrying drugs toward the target etc.
Biomaterials play an integral role in medicine today—restoring function and facilitating healing for people after injury or disease. Biomaterials may be natural or synthetic and are used in medical applications to support, enhance, or replace damaged tissue or a biological function. The first historical use of biomaterials dates to antiquity, when ancient Egyptians used sutures made from animal sinew. The modern field of biomaterials combines medicine, biology, physics, and chemistry, and more recent influences from tissue engineering and materials science. The field has grown significantly in the past decade due to discoveries in tissue engineering, regenerative medicine, and more.
Metals, ceramics, plastic, glass, and even living cells and tissue all can be used in creating a biomaterial. They can be reengineered into molded or machined parts, coatings, fibers, films, foams, and fabrics for use in biomedical products and devices.
Finally, doctors and scienticts in general, use biomaterials because they have a broad range of applications such as, joint replacements, drug delivery mechanis, msartificial ligaments and tendons etc. [2]

2 The Fuse Model

As we know models are an essential part of physics. One of the most well-known models is the fuse model, which has been the statistical physics community's workhorse for fracture phenomena, since 1985.
The fuse model consists of a lattice in which each bond is an electrical fuse, so we can say that it behaves as an ohmic resistor up to a maximum current. If this threshold t is exceeded, the bond "burns out" and its conductance drops to zero. Moreover, the thresholds are drawn from a statistical distribution that's usually chosen to be spatially uncorrelated. We then drive an increasing current through the lattice and record the successive breakdown of the fuses. The fuse network starts with all fuses intact, in which we will then gradually increase the voltage V in the network until the fuse with the lowest threshold t is blown.In this way the current is instantaneously redistributed in the network, so we will then proceed to a second voltage increase (gradually) until a second fuse is blown, and so on.

3 Our Random Fuse Model

To study our random fuse model problem, we used the jupyter notebook which contains the implementation of our model in a square lattice, similar to that introduced in the CoMBiF course. However, the system is subject to a uniaxial load in the vertical direction, with constant voltages $V = 1$ and $V = 0$ applied to the top and bottom boundary respectively. Also, periodic boundary conditions are applied in the horizontal direction. This configuration mimics the geometry of an elastic/brittle material, which is subject to tension in the vertical direction. In the current implementation, four contiguous vertical links are removed, in order to simulate a horizontal crack. The two figures in the notebook show i) the actual fuse network structure and ii) the currents flowing in the system as a consequence of the applied load and the crack, where we represent higher currents with thicker lines, as we can see also in the figures 1 and 2. Furthermore, because of the crack, some currents flow in the horizontal direction, mimicking load redistribution around the crack profile, in analogy with patterns of stress redistribution in fracture mechanics.

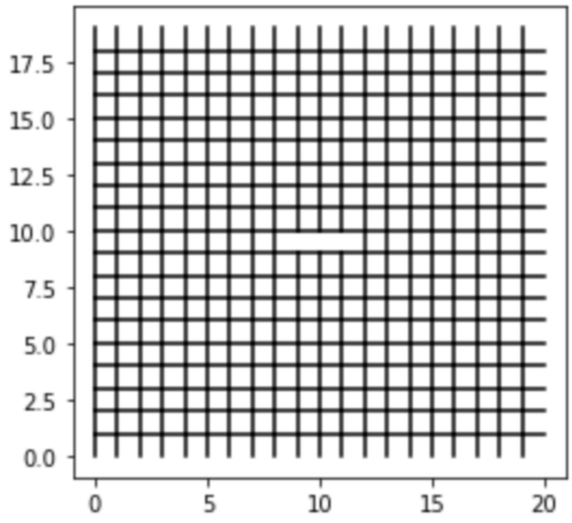

Figure 1: The actual fuse network structure

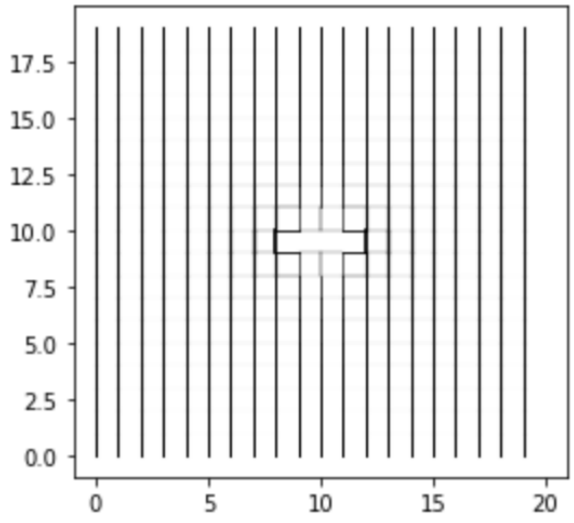

Figure 2: The currents flowing in the system as a consequence of the applied load and the crack

After we familiarized ourselves with the code and according the videos of the course CoMBiF, we studied the dependence of I on the length of the crack a. As it mentioned, the global current I, that carried by the system (in simulation units) is the current carried by all vertical links in any horizontal cross-section of the network. The results are shown below in the Table 1 and the images 3,4 and 5 are shown the network, the currents and the potential of our system respectively.

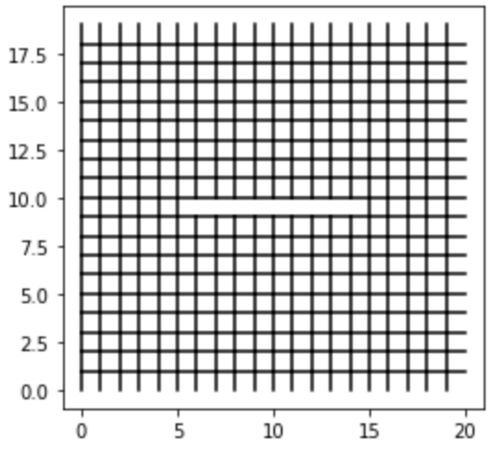

Figure 3: The global network of the system.

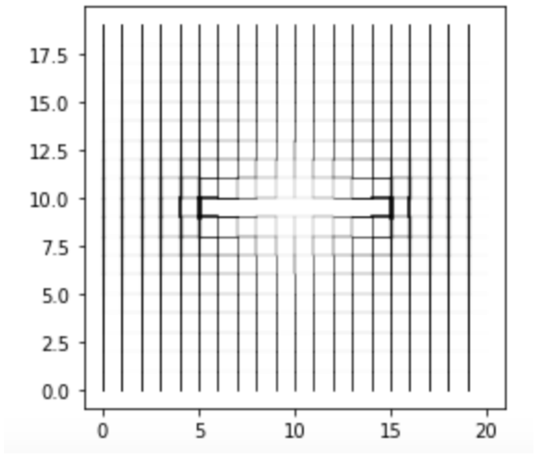

Figure 4: The global current carried by the system.

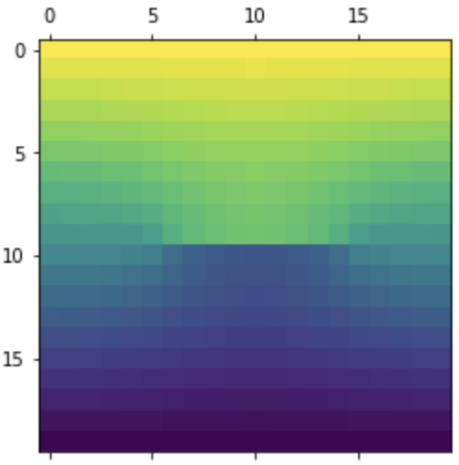

Figure 5: The potential of the system.

Also, as we know the currents in horizontal links are higher near the crack and decrease with the distance from the crack (for instance, in the direction normal to the crack line). To prove that, we fixed a crack size and then we studied the dependence of these currents on the distance from the crack. The results that we found are shown in the table 2.

crack	length
a=0,	l= 1.052631578947382
a=1,	l= 1.0471069276397802
a=2,	l= 1.0375018632650965
a=3,	l= 1.0238083111977723
a=4,	l= 1.0061556933254212
a=5,	l= 0.9847336605192087
a=6,	l= 0.9597633410169697
a=7,	l= 0.9314801706137471
a=8,	l= 0.9001193246000948
a=9,	l= 0.8659013783211617

Table 1: The values of the cracks and the corresponding lengths.

horizontal currents	distance
193	0.05819824788614636
194	0.01573200592443269,
195	0.005177364271676033
196	0.0021004413305583114
197	0.0010041085274388184
198	0.0005201346880744362
199	0.00025661551570665964

Table 2: The horizontal currents and the corresponding distances

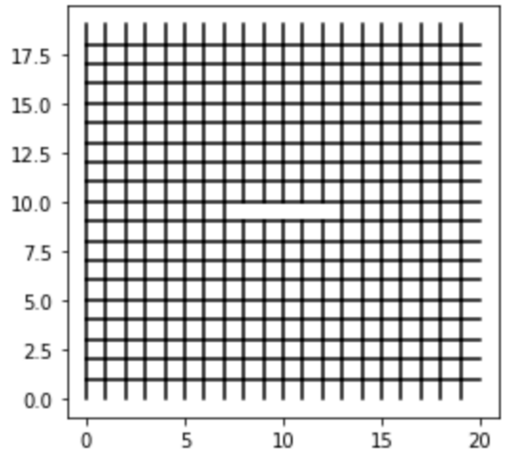

Figure 6: The network of the system.

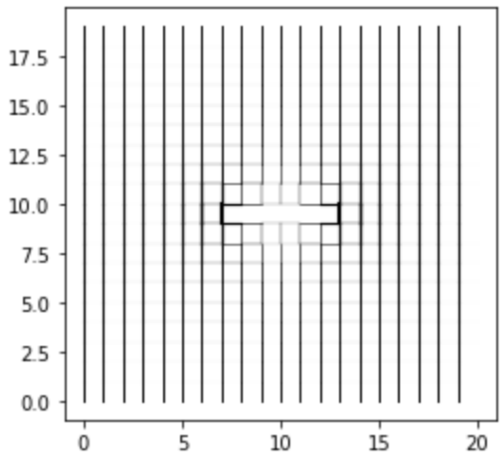

Figure 7: The currents in horizontal links.

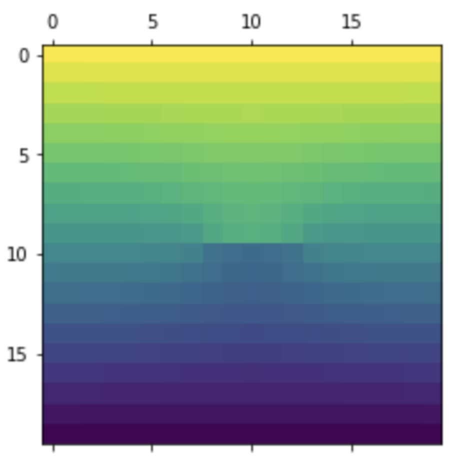

Figure 8: The potential of the system.

4 Conclusion

To sum up, from the results above we can understand that as it concerns the random fuse model, the global current I carried by the system (in simulation units) is the current carried by all vertical links in any horizontal cross-section of the network. At this case, we studied the dependence of I on the length of the crack a and we ended up that the results show little difference. Also, as we can observe in Figures 6-8, from the constant crack size and the study of the dependence of these currents on the distance from the crack, the distance decreases as we move away from the crack.

5 Appendix

5.1 Main Program

```
from ˙˙future˙˙ import division
from ˙˙future˙˙ import print˙function
import numpy as np
import scipy.sparse as ssp
from scipy.sparse.linalg import spsolve
import matplotlib.pyplot as plt

class FiberNetwork:
    def ˙˙init˙˙(self,Nx,Ny):
        self.Nx = Nx
        self.Ny = Ny
        self.N = Nx*Ny
        self.nrows = self.N - 2*self.Nx
        self.adjmat = ssp.lil˙matrix((self.N,self.N), dtype=np.int32)
        self.V = np.zeros(self.N, dtype=np.float64)
        self.V[:Nx] = 1
    def adj˙todense(self):
        return self.adjmat.todense()
    def lap˙todense(self):
        return self.lapmat.todense()
    def get˙east˙neighbour(self,node):
        return (node//self.Nx)*self.Nx + (node+1)%self.Nx
    def get˙south˙neighbour(self,node):
        return (node+self.Nx)%self.N
    def add˙link(self,node1,node2):
        self.adjmat[node1,node2] = 1
        self.adjmat[node2,node1] = 1
    def build˙fiber˙network˙w˙bus(self):
        for node in range(self.Nx, self.N-self.Nx):
            self.add˙link(node,self.get˙east˙neighbour(node))
        for node in range(self.N-self.Nx):
            self.add˙link(node,self.get˙south˙neighbour(node))
    def get˙neighbors(self,node):
        return self.adjmat.rows[node]
    def lapmat(self):
        lapmat = ssp.lil˙matrix((self.nrows,self.nrows),
        ↪   dtype=np.float64)
        for node in range(self.Nx, self.N-self.Nx):
            i = node - self.Nx
            neighbors = self.get˙neighbors(node)
            degree = len(neighbors)
            lapmat[i,i] = degree
            for nn in neighbors:
                if(nn¿=self.Nx and nn¡self.N-self.Nx):
                    j = nn - self.Nx
                    lapmat[i,j] = -1
        return lapmat.tocsr()
    def rhs(self):
        rhs = np.zeros(self.nrows, dtype=np.float64)
        for node in range(self.Nx, 2*self.Nx):
            neighbors = self.get˙neighbors(node)
            if node-self.Nx in neighbors:
                rhs[node-self.Nx] = self.V[node-self.Nx]
        return rhs
    def update˙potentials(self,x):
        self.V[self.Nx:self.N-self.Nx] = x
    def remove˙fiber˙between˙nodes(self,node1,node2):
        self.adjmat[node1,node2] = 0
        self.adjmat[node2,node1] = 0
    def plot˙potentials(self):
        side = self.Nx
        matrixV = self.V.reshape((side,side))
        plt.matshow(matrixV)
    def plot˙network(self):
        f, ax = plt.subplots()
        ax.set˙aspect('equal')
        elements = self.adjmat.nonzero()
        for i,j in zip(elements[0],elements[1]):
            if i¡j:
                xi, xj = i%self.Nx, j%self.Nx
                yi, yj = i//self.Nx, j//self.Nx
                if j-i==self.Nx-1:
                    xi = xj+1
                ax.plot([xi,xj],[yi,yj],'k-')
    def plot˙currents(self):
        f, ax = plt.subplots()
        ax.set˙aspect('equal')
        elements = self.adjmat.nonzero()
        for i,j in zip(elements[0],elements[1]):
            if i¡j:
                xi, xj = i%self.Nx, j%self.Nx
                yi, yj = i//self.Nx, j//self.Nx
                current = np.abs(self.V[i]-self.V[j])
                if j-i==self.Nx-1:
                    xi = xj+1
                ax.plot([xi,xj],[yi,yj],'k',lw=20*current)

net = FiberNetwork(20,20)
net.build˙fiber˙network˙w˙bus()
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)

net.remove˙fiber˙between˙nodes(189,209)
net.remove˙fiber˙between˙nodes(190,210)
net.remove˙fiber˙between˙nodes(191,211)
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)

net.plot˙network()
net.plot˙currents()
plt.show()
```

5.2 Task 2

```
from ˙˙future˙˙ import division
from ˙˙future˙˙ import print˙function
import numpy as np
import scipy.sparse as ssp
from scipy.sparse.linalg import spsolve
import matplotlib.pyplot as plt

class FiberNetwork:
    def ˙˙init˙˙(self,Nx,Ny):
        self.Nx = Nx
        self.Ny = Ny
        self.N = Nx*Ny
        self.nrows = self.N - 2*self.Nx
        self.adjmat = ssp.lil˙matrix((self.N,self.N), dtype=np.int32)
        self.V = np.zeros(self.N, dtype=np.float64)
        self.V[:Nx] = 1
    def adj˙todense(self):
        return self.adjmat.todense()
    def lap˙todense(self):
        return self.lapmat.todense()
    def get˙east˙neighbour(self,node):
        return (node//self.Nx)*self.Nx + (node+1)%self.Nx
    def get˙south˙neighbour(self,node):
        return (node+self.Nx)%self.N
    def add˙link(self,node1,node2):
        self.adjmat[node1,node2] = 1
        self.adjmat[node2,node1] = 1
    def build˙fiber˙network˙w˙bus(self):
        for node in range(self.Nx, self.N-self.Nx):
            self.add˙link(node,self.get˙east˙neighbour(node))
        for node in range(self.N-self.Nx):
            self.add˙link(node,self.get˙south˙neighbour(node))
    def get˙neighbors(self,node):
        return self.adjmat.rows[node]
    def lapmat(self):
        lapmat = ssp.lil˙matrix((self.nrows,self.nrows),
        ↪   dtype=np.float64)
        for node in range(self.Nx, self.N-self.Nx):
            i = node - self.Nx
            neighbors = self.get˙neighbors(node)
            degree = len(neighbors)
            lapmat[i,i] = degree
            for nn in neighbors:
                if(nn¿=self.Nx and nn¡self.N-self.Nx):
                    j = nn - self.Nx
                    lapmat[i,j] = -1
        return lapmat.tocsr()
    def rhs(self):
        rhs = np.zeros(self.nrows, dtype=np.float64)
        for node in range(self.Nx, 2*self.Nx):
            neighbors = self.get˙neighbors(node)
            if node-self.Nx in neighbors:
                rhs[node-self.Nx] = self.V[node-self.Nx]
        return rhs
    def update˙potentials(self,x):
        self.V[self.Nx:self.N-self.Nx] = x
    def remove˙fiber˙between˙nodes(self,node1,node2):
        self.adjmat[node1,node2] = 0
        self.adjmat[node2,node1] = 0
    def plot˙potentials(self):
        side = self.Nx
        matrixV = self.V.reshape((side,side))
        plt.matshow(matrixV)
    def plot˙network(self):
        f, ax = plt.subplots()
        ax.set˙aspect('equal')
        elements = self.adjmat.nonzero()
        for i,j in zip(elements[0],elements[1]):
            if i¡j:
                xi, xj = i%self.Nx, j%self.Nx
                yi, yj = i//self.Nx, j//self.Nx
                if j-i==self.Nx-1:
                    xi = xj+1
                ax.plot([xi,xj],[yi,yj],'k-')
    def plot˙currents(self):
        f, ax = plt.subplots()
        ax.set˙aspect('equal')
        elements = self.adjmat.nonzero()
        for i,j in zip(elements[0],elements[1]):
            if i¡j:
                xi, xj = i%self.Nx, j%self.Nx
                yi, yj = i//self.Nx, j//self.Nx
                current = np.abs(self.V[i]-self.V[j])
                if j-i==self.Nx-1:
                    xi = xj+1
                ax.plot([xi,xj],[yi,yj],'k-',lw=20*current)
    def global˙current(self):
        globcurrent = 0.
        for i in range(self.Nx):
            current = np.abs(self.V[i]-self.V[i+self.Nx])
            globcurrent += current
        return globcurrent

net = FiberNetwork(20,20)
net.build˙fiber˙network˙w˙bus()

x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)
print('a=0,  I=',net.global˙current())

net.remove˙fiber˙between˙nodes(190,210)
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)
print('a=1,  I=',net.global˙current())

net.remove˙fiber˙between˙nodes(189,209)
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)
print('a=2,  I=',net.global˙current())

net.remove˙fiber˙between˙nodes(191,211)
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)
print('a=3,  I=',net.global˙current())

net.remove˙fiber˙between˙nodes(188,208)
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)
print('a=4,  I=',net.global˙current())

net.remove˙fiber˙between˙nodes(192,212)
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)
print('a=5,  I=',net.global˙current())

net.remove˙fiber˙between˙nodes(187,207)
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)
print('a=6,  I=',net.global˙current())

net.remove˙fiber˙between˙nodes(193,213)
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)
print('a=7,  I=',net.global˙current())

net.remove˙fiber˙between˙nodes(186,206)
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)
print('a=8,  I=',net.global˙current())

net.remove˙fiber˙between˙nodes(194,214)
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)
print('a=9,  I=',net.global˙current())


net.plot˙network()
net.plot˙currents()
net.plot˙potentials()
plt.show()
```

5.3 Task 3

```
from ˙˙future˙˙ import division
from ˙˙future˙˙ import print˙function
import numpy as np
import scipy.sparse as ssp
from scipy.sparse.linalg import spsolve
import matplotlib.pyplot as plt

class FiberNetwork:
    def ˙˙init˙˙(self,Nx,Ny):
        self.Nx = Nx
        self.Ny = Ny
        self.N = Nx*Ny
        self.nrows = self.N - 2*self.Nx
        self.adjmat = ssp.lil˙matrix((self.N,self.N), dtype=np.int32)
        self.V = np.zeros(self.N, dtype=np.float64)
        self.V[:Nx] = 1
    def adj˙todense(self):
        return self.adjmat.todense()
    def lap˙todense(self):
        return self.lapmat.todense()
    def get˙east˙neighbour(self,node):
        return (node//self.Nx)*self.Nx + (node+1)%self.Nx
    def get˙south˙neighbour(self,node):
        return (node+self.Nx)%self.N
    def add˙link(self,node1,node2):
        self.adjmat[node1,node2] = 1
        self.adjmat[node2,node1] = 1
    def build˙fiber˙network˙w˙bus(self):
        for node in range(self.Nx, self.N-self.Nx):
            self.add˙link(node,self.get˙east˙neighbour(node))
        for node in range(self.N-self.Nx):
            self.add˙link(node,self.get˙south˙neighbour(node))
    def get˙neighbors(self,node):
        return self.adjmat.rows[node]
    def lapmat(self):
        lapmat = ssp.lil˙matrix((self.nrows,self.nrows),
        ↪   dtype=np.float64)
        for node in range(self.Nx, self.N-self.Nx):
            i = node - self.Nx
            neighbors = self.get˙neighbors(node)
            degree = len(neighbors)
            lapmat[i,i] = degree
            for nn in neighbors:
                if(nn¿=self.Nx and nn¡self.N-self.Nx):
                    j = nn - self.Nx
                    lapmat[i,j] = -1
        return lapmat.tocsr()
    def rhs(self):
        rhs = np.zeros(self.nrows, dtype=np.float64)
        for node in range(self.Nx, 2*self.Nx):
            neighbors = self.get˙neighbors(node)
            if node-self.Nx in neighbors:
                rhs[node-self.Nx] = self.V[node-self.Nx]
        return rhs
    def update˙potentials(self,x):
        self.V[self.Nx:self.N-self.Nx] = x
    def remove˙fiber˙between˙nodes(self,node1,node2):
        self.adjmat[node1,node2] = 0
        self.adjmat[node2,node1] = 0
    def plot˙potentials(self):
        side = self.Nx
        matrixV = self.V.reshape((side,side))
        plt.matshow(matrixV)
    def plot˙network(self):
        f, ax = plt.subplots()
        ax.set˙aspect('equal')
        elements = self.adjmat.nonzero()
        for i,j in zip(elements[0],elements[1]):
            if i¡j:
                xi, xj = i%self.Nx, j%self.Nx
                yi, yj = i//self.Nx, j//self.Nx
                if j-i==self.Nx-1:
                    xi = xj+1
                ax.plot([xi,xj],[yi,yj],'k-')
    def plot˙currents(self):
        f, ax = plt.subplots()
        ax.set˙aspect('equal')
        elements = self.adjmat.nonzero()
        for i,j in zip(elements[0],elements[1]):
            if i¡j:
                xi, xj = i%self.Nx, j%self.Nx
                yi, yj = i//self.Nx, j//self.Nx
                current = np.abs(self.V[i]-self.V[j])
                if j-i==self.Nx-1:
                    xi = xj+1
                ax.plot([xi,xj],[yi,yj],'k-',lw=20*current)
    def hor˙current(self,i):
        hor˙current = []
        for j in range((i//self.Nx+1)*self.Nx - i-1):
            print(i+j+1)
            hor˙current.append(np.abs(self.V[i+j]-self.V[i+j+1]))
        return hor˙current

net = FiberNetwork(20,20)
net.build˙fiber˙network˙w˙bus()

x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)

net.remove˙fiber˙between˙nodes(188,208)
net.remove˙fiber˙between˙nodes(189,209)
net.remove˙fiber˙between˙nodes(190,210)
net.remove˙fiber˙between˙nodes(191,211)
net.remove˙fiber˙between˙nodes(192,212)
x = spsolve(net.lapmat(), net.rhs())
net.update˙potentials(x)

print(net.hor˙current(192))

net.plot˙network()
net.plot˙currents()
net.plot˙potentials()
plt.show()
```

References

[1] “Materials science - wikipedia,” Aug. 2022.

[2] “Biomaterials,” Aug. 2022.