Math574 - Homework6

Altansuren Tumurbaatar

Spring 2016

1. Construct two topological spaces that have isomorphic homology groups, but are not homotopy equivalent.

   Let K_1 and K_2 be a simplicial complexes as illustrated in Figure1. Both complexes are orientable and have the same topological invariants such as betti numbers and Euler characteristic. Hence, K_1 and K_2 have the same homology groups as follows. For $i = 1, 2$ $\beta_0(K_i) = 2$
   $\beta_1(K_i) = 3$
   $\beta_2(K_i) = 0$
   $\chi(K_i) = 2 - 3 + 0 = -1$
   While,K_1 and K_2 are not homotopy equivalent since there is no continuous map between them since triangle is homologous to a circle and a circle is not contractible. In K_1, the isolates triangle is not contractible to a point and all the other triangles are also not contractible to a point while K_2 has a point.

2. Recall that $K^{(d)}$ denotes the d-skeleton of the simplicial complex K, which contains all simplices of K with dimension d or smaller. For what values of p is it true that $H_p(K^{(d)}) \cong H_p(K)$?

   - When $d = dim(K)$:
     It is trivial and $H_p(K^{(d)}) \cong H_p(K) \forall p$

   - When $d < dim(K)$:
     $K^{(d)}$ includes equal to d or lower dimensional simplices.
     $\Rightarrow Z_p(K^{(d)}) = Z_p(K)$, $\forall p <= d$ and
     $B_p(K^{(d)}) = B_p(K)$, $\forall p < d$ by boundary homomorphism.
     Furthermore,
     when p=d, $B_p(K^{(d)}) = 0$ and $B_p(K^{(d)}) = B_p(K) if B_p(K) = 0$

     $\Rightarrow$ Thus, $H_p(K^{(d)}) \cong H_p(K)$, $\forall p < d$, and $H_p(K^{(d)}) \cong H_p(K)$, for $p = d$ if $B_d(K) = 0$,

3. Find the Betti numbers β_0, β_1 and β_2 of the Klein bottle over both $\mathbb{Z}_2$ and $\mathbb{Z}$. You could use the triangulation of the Klein bottle introduced in Lecture 6. Which other 2-manifolds introduced in class have the same set of Betti numbers ($\mathbb{Z}_2$ and $\mathbb{Z}$)? (You are welcome to use the code in the following question for this problem as well, where applicable.)

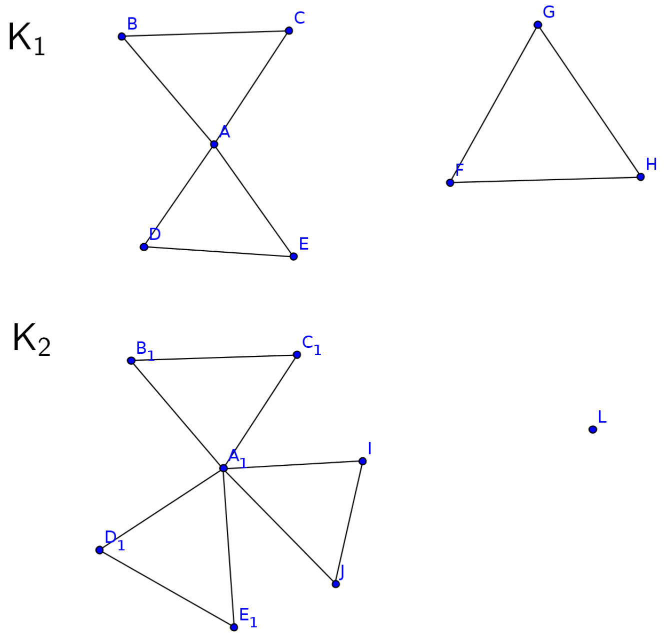

Figure 1:

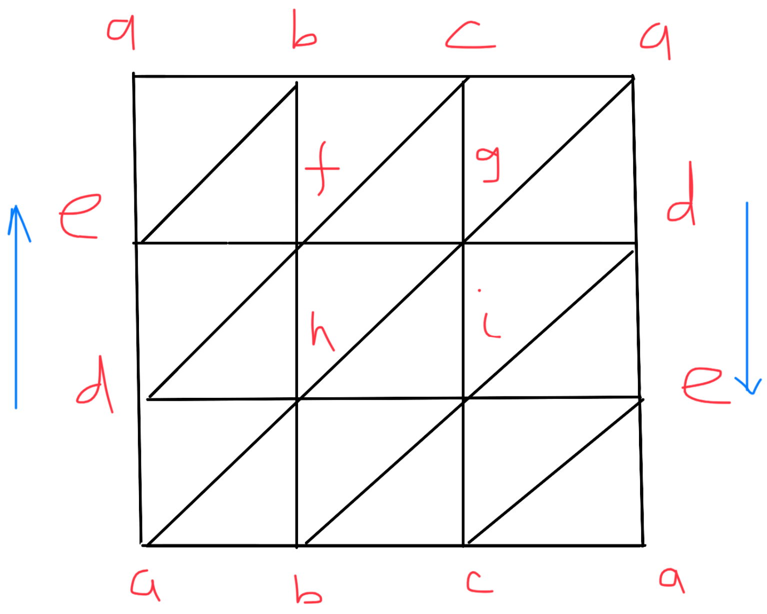

Figure 2: Triangulation of $\mathbb{K}^2$. All triangles are filled.

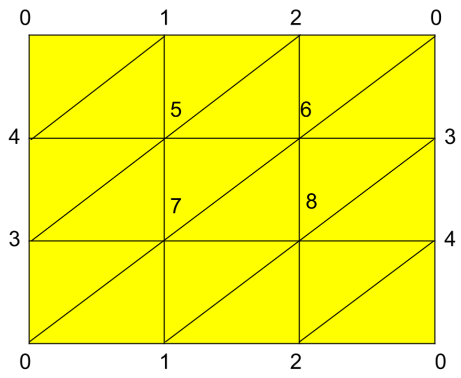

Figure 3: Relabeled vertices of $\mathbb{K}^2$

See triangulation of $\mathbb{K}^2$ from Figure 2

Betti numbers of Klein bottle over $\mathbb{Z}$.

- Number of connected components = 1:
  $H_0(\mathbb{K}^2) = 1$
  $\Rightarrow \beta_0 = 1$
- Number of holes(cycles) = 1:
  Let $\bar{w} = [a,b] + [b,c] + [c,a]$ be a 1-cycle going around horizontally and $\bar{z} = [a,d] + [d,e] + [e,a]$ be another 1-cycle going around vertically. Then $2\bar{z}$ generates B_2 and $\forall n \in Z$, $2n\bar{z}$ is 1-boundary. Thus, $H_1(\mathbb{K}^2) = \mathbb{Z} \oplus \mathbb{Z}_2$ where $\mathbb{Z}_2$ is a torsion part and $\mathbb{Z}$ is for β_1. There is one torsion-free 1-cycle in $\mathbb{K}^2$ which is $\bar{w}$.
  $\Rightarrow \beta_1 = 1$
- Number of enclosed voids = 0:
  We know, $\partial_2 = 2\bar(z) \Rightarrow Z_2(\mathbb{K}^2) = 0$
  There is no 3-simplices in $\mathbb{K}^2 \Rightarrow B_3(\mathbb{K}^2) = 0$
  Hence, $H_2(\mathbb{K}^2) = 0$ and there is no enclosed void in it.
  $\Rightarrow \beta_2 = 0$
- $\forall \beta_i, i > 2$ are trivial.

Betti numbers of Klein bottle over $\mathbb{Z}_2$.

- Number of connected components = 1:
  $\Rightarrow \beta_0 = 1$
- Number of holes(cycles) = 2:
  Let $\bar{w} = [a,b]+[b,c]+[c,a]$ be a 1-cycle going around horizontally and $\bar{z} = [a,d] + [d,e] + [e,a]$ be another 1-cycle going around vertically. Then $2\bar{z}$ generates B_2 and $\forall n \in \mathbb{Z}$, $2n\bar{z}$ is 1-boundary. Since we are computing over $\mathbb{Z}_2$, all the 1-boundaries, $\forall n \in \mathbb{Z}$, $2n\bar{z}$ are removed and $(2n + 1) * \bar{z}$ is counted as 1-cycle. $\Rightarrow \beta_1 = 2$
- Number of enclosed voids = 0:
  We know, $\partial_2(\mathbb{K}^2) = 2\bar(z)\%2 = 0 \Rightarrow Z_2(\mathbb{K}^2) = 1$

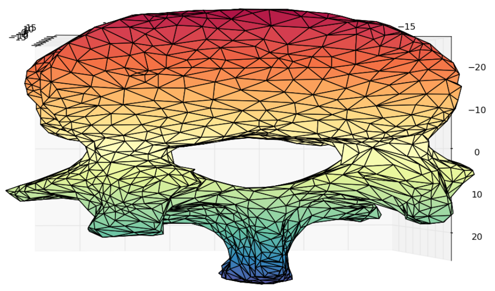

Figure 4: The spine complex

There is no 3-simplices in $\mathbb{K}^2 \Rightarrow B_3(\mathbb{K}^2) = 0$
Hence, $H_2(\mathbb{K}^2) = 1$ over $\mathbb{Z}_2$
$\Rightarrow \beta_2 = 1$

- $\forall \beta_i, i > 2$ are trivial.

A klein bottle in $\mathbb{Z}_2$ has the same β_0, β_1 and β_2 as a torus in $\mathbb{Z}$

Outputs are appended in the end.

4. Write a function in Octave or another similar package/language that takes as input a matrix with entries in $0, 1$, and finds its Smith normal form (SNF) when working over Z_2. Use the above function to compute all relevant Betti numbers of the patched triangu- lation of the spine seen in Homework 3, i.e., after you closed the holes by adding three triangles.

   The spine complex(See Figure4)is homomorphic to torus and torus has β_0, β_1,β_2 as 1,2,1 respectively computed over $\mathbb{Z}$, so does the spine complex. The betti numbers of the spine complex computed over $\mathbb{Z}_2$ such as β_0, β_1,β_2 are also 1,2,1. The reason is why they match is that torus is orientable 2-manifold. For orientable 2-manifold, its betta numbers in $\mathbb{Z}$ are the same as its betta numbers in $\mathbb{Z}_2$. Matrix reduction of the two boundary matrices takes around 30 minutes total. I computed them once and read them from files. Source code and outputs are appended next page.

```
'''
Math574 - Computationa Topology - Spring 2016
---------------------------------------------

Homework6 on Smith Normal Form(SNF).

*Tasks:*
1. Given a boundary matrix, compute SNF over Z/2
   Refer to a function called SNF()
   - Used recursive matrix reduction algorithm
       Algorithm steps are commented in the function
       This algorithm does reduction recursively and for each recursion
       it works on smaller submatrix. The submatrix sizes decrease as follows.
       [mxn],[(m-1)x(n-1)], [(m-2)x(n-2)], ... ,[1,1].
       It interrupts if there is no nonzero element at the indices
       greater or equal to current iteration. On the other words, if the current
       submatrix is zero matrix, it stops.
   - Data structure - Sparse dok_matrix which support index lookup,
     assignment and simple operations such as multiplication, addition.
     Besides, it supports dot product. Though it doesn't support
     module(%) operation. Hence, +/2 is done separately on nonzero elements
     after every addition of row or column. Since the number of nonzeros are small,
     it doesn't increase the complexity of the algorithm much.
2. Find all relevant Betti numbers of the triangulation of the spine given
    Triangles.txt, Vertices.txt
3.Plot the surface

*Data:*
Triagles.txt - List of triangles(mesh) which approxiates the surface
Vertices.txt - Coordinates of the points used in the trianglular mesh
'''
import os
import time
import numpy as np
from scipy.sparse import dok_matrix, find, identity
from scipy.sparse.linalg import inv
import scipy.sparse as sparse

from mpl_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import sys
sys.setrecursionlimit(10**6)

N = None
U = None
V = None

def get_subsimplices(simplices):
    simplices = np.sort(simplices, axis=1)
    subsimplices = set()
    for j in np.arange(simplices.shape[1]):
        idx = list(range(simplices.shape[1]))
        idx.pop(j)
        subsimplices = subsimplices.union(set(tuple(sub) for sub in simplices.take(idx
, axis=1)))
    subsimplices = np.array([ sub for sub in subsimplices], dtype=int)
    return subsimplices

def boundary_matrix(simplices, subsimplices, is_sparse=True, format='coo'):
    simplex_dim  = simplices.shape[1]
    n_simplices = simplices.shape[0]
    m_subsimplices = subsimplices.shape[0]
    if is_sparse:
        boundary_matrix = dok_matrix((m_subsimplices, n_simplices), dtype=int)
    else:
        boundary_matrix = np.array((m_subsimplices, n_simplices), dtype=int)
    val = 1
    simplices = np.sort(simplices, axis=1)
    subsimplices = np.sort(subsimplices, axis=1)
    for i, simplex in enumerate(simplices):
        for j in np.arange(simplex_dim):
            idx = list(range(simplex_dim))
            idx.pop(j)
            subsimplex = simplex.take(idx)
            # to check the membership of subsimplex in subsimplices
            subsimplex_idx = np.argwhere((subsimplices==subsimplex).all(axis=1) == Tru
e)
            if subsimplex_idx.size == 0:
                sys.stderr.write("Unable to find subsimplex! Make sure subsimplices co
ntains all boundary subsimplices\n")
                exit()
            subsimplex_idx = subsimplex_idx[0][0]
            boundary_matrix[subsimplex_idx, i] = val
    if is_sparse:
        return boundary_matrix.asformat(format)
    else:
        return boundary_matrix

def SNF(x, uv=False):
    #print x
    global N
    global U
    global V
    #Get indices of nonzeros
    tmp = find(N)
    nonzero = tmp[0].reshape(-1,1)
    nonzero = np.hstack((nonzero, tmp[1].reshape(-1,1)))
    if np.amax(nonzero) < x:
        return
    #Get indices of nonzeros which is greater or equal to x
    idx = np.argwhere(np.all(nonzero>=x, axis=1) == True).reshape(-1,)
    idx = idx[idx.argsort()]
    if len(idx) !=0:
        if N[x,x] == 0:
            #Choose the least indices of nonzeros which is greater or equal to x
            k, l = nonzero[idx[0]]
            if k != x:
                #Exchanging xth row wih kth row
                x_nonzero = N[x, :].nonzero()[1]
                k_nonzero = N[k,:].nonzero()[1]
                for j in x_nonzero:
                    N[x,j] = 0
                for j in k_nonzero:
                    N[x,j] = 1
                    N[k,j] = 0
                for j in x_nonzero:
                    N[k,j] = 1
                if uv:
                    # Update U
                    Utmp = dok_matrix(identity(N.shape[0], dtype=int), dtype=int)
                    Utmp[k,x] = 1
                    Utmp[x,k] = 1
                    Utmp[k,k] = 0
                    Utmp[x,x] = 0
                    U = (Utmp*U).asformat("dok").astype(int)
                    del Utmp
                    for key,val in U.items():
                        U[key[0], key[1]] = val%2
            if l != x:
                #Exchanging xth column wih lth column
                x_nonzero = N[:, x].nonzero()[0]
                l_nonzero = N[:, l].nonzero()[0]
                for i in x_nonzero:
                    N[i, x] = 0
                for i in l_nonzero:
                    N[i,x] = 1
                    N[i,l] = 0
                for i in x_nonzero:
                    N[i,l] = 1
                if uv:
                    # Update V
                    Vtmp = dok_matrix(identity(N.shape[1], dtype=int), dtype=int)
                    Vtmp[l,x] = 1
                    Vtmp[x,l] = 1
                    Vtmp[l,l] = 0
                    Vtmp[x,x] = 0
                    V = (V*Vtmp).asformat("dok").astype(int)
                    del Vtmp
                    for key,val in V.items():
                        V[key[0], key[1]] = val%2
        #Iterate over rows of N down xth row
        x_col = N[:,x].nonzero()[0]
        x_col_i = x_col[np.argwhere(x_col >= x+1).reshape(-1,)]
        x_col_i = np.sort(x_col_i)
        x_row = N[x,:].nonzero()[0]
        x_row_j = x_row[np.argwhere(x_row >= x+1).reshape(-1,)]
        x_row_j = np.sort(x_row_j)
        for i in x_col_i:
            # Add row x to row i
            N[i,:] = N[i,:] + N[x,:]
            for key, val in N[i,:].items():
                N[i, key[1]] = val%2
            if uv:
                # Update U
                Utmp = dok_matrix(identity(N.shape[0], dtype=int), dtype=int)
                Utmp[i,x] = 1
                U = (Utmp*U).asformat("dok").astype(int)
                del Utmp
                for key,val in U.items():
                    U[key[0], key[1]] = val%2
        #Iterate over colmns of N right to col x
        x_row = N[x,:].nonzero()[1]
        x_row_j = x_row[np.argwhere(x_row >= x+1).reshape(-1,)]
        x_row_j = np.sort(x_row_j)
        for j in x_row_j:
            # Add col x to col j
            N[:,j] = N[:,j] + N[:,x]
            for key, val in N[:,j].items():
                N[key[0], j] = val%2
            if uv:
                # Update V
                Vtmp = dok_matrix(identity(N.shape[1], dtype=int), dtype=int)
                Vtmp[x,j] = 1
                V = (V*Vtmp).asformat("dok").astype(int)
                del Vtmp
                for key,val in V.items():
                    V[key[0], key[1]] = val%2
        SNF(x+1)

# Saves sparse matrix as text. if the input is not sparse, set is_sparse argument to F
alse.
def sparse_savetxt(fname, matrix, fmt='%d', include_dim=False):
    if sparse.issparse(matrix):
        if matrix.getformat() !='coo':
            matrix = matrix.asformat('coo')
    else:
        matrix = sparse.coo_matrix(matrix, dtype=matrix.dtype)
    with open(fname, 'w') as f:
        if include_dim:
            f.write("%d %d \n" % (matrix.shape[0], matrix.shape[1]))
        for i in range(len(matrix.row)):
            fmt_str = "%d %d " + fmt + "\n"
            f.write(fmt_str % (matrix.row[i], matrix.col[i], matrix.data[i]))
#Loads SNF matrix from text file and returns in dok_matrix format
```

```
def load_N(fname):
    N = None
    if os.path.exists(fname):
        with open(fname, 'r') as f:
            for i, line in enumerate(f.readlines()):
                if i == 0:
                    data = line.split()
                    N = sparse.dok_matrix((int(data[0]), int(data[1])), dtype=int)
                else:
                    data = line.split()
                    N[int(data[0]), int(data[1])] = int(data[2])
    else:
        print "Can't load <N>. <%s> doesn't exist"%fname
    return N

def spinedemo():
    global N
    global U
    global V

    start = time.time()
    triangles = np.loadtxt("Triangles.txt")
    triangles = triangles -1
    triangles = np.vstack((triangles, np.array([498,499,501]).reshape(1,-1)))
    triangles = np.vstack((triangles, np.array([341,348,1256]).reshape(1,-1)))
    triangles = np.vstack((triangles, np.array([90,94,263]).reshape(1,-1)))
    points = np.loadtxt("Vertices.txt")
    edges = get_subsimplices(triangles)
    vertices = get_subsimplices(edges)
    b_matrix2 = boundary_matrix(triangles, edges, format="dok")
    b_matrix1 = boundary_matrix(edges, np.arange(0, len(points)).reshape(-1,1), format
="dok")

    B = b_matrix2
    N = dok_matrix(B, dtype=int)
    U = dok_matrix(identity(N.shape[0], dtype=int), dtype=int)
    V = dok_matrix(identity(N.shape[1], dtype=int), dtype=int)
    # Smith Normal Form
    #SNF(0, uv=False)
    #sparse_savetxt("spineSNF2.txt", N, fmt="%d", include_dim=True)
    N = load_N("spineSNF2.txt")

    # Boundary and cycle ranks
    b2 = 0
    b1 = np.amax(N.nonzero()[1])+1
    z2 = N.shape[1] - b1

    B = b_matrix1
    N = dok_matrix(B, dtype=int)
    U = dok_matrix(identity(N.shape[0], dtype=int), dtype=int)
    V = dok_matrix(identity(N.shape[1], dtype=int), dtype=int)
    # Smith Normal Form
    #SNF(0, uv=False)
    #sparse_savetxt("spineSNF1.txt", N, fmt="%d", include_dim=True)
    N = load_N("spineSNF1.txt")

   # Boundary and cycle ranks
    b0 = np.amax(N.nonzero()[1])+1
    z1 = N.shape[1] - b0
    # There are 4 isolated points in K which should be ignored.
    z0 = N.shape[0] -4 #Number of vertices
    print "==========================================================="
    print "Spine"
    print "==========================================================="
    print "Boundary matrix dimension(EDGESxTRIANGLES):%dx%d"%(b_matrix2.shape)
    print "Boundary matrix dimension(VERTICESxEDGES):%dx%d"%(b_matrix1.shape[0]-4, b_m
atrix1.shape[1])
    print r"Rank of B2: ", b2
    print r"Rank of Z2: ", z2
    print r"Rank of B1: ", b1
    print r"Rank of Z1: ", z1
    print r"Rank of B0: ", b0
    print r"Rank of Z0: ", z0
    print "Betti 0 = (z0-b0): ", z0 - b0
    print "Betti 1 = (z1-b1): ", z1-b1
    print "Betti 2 = (z2-b2):", z2-b2
    print "Betti i, i>2 is trivial since there is no tetrahedra in K"
    elapsed = time.time() - start
    print "Time(min) spent: %f" % (elapsed/60)
    '''
    #Plotting
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1, projection='3d')
    ax.plot_trisurf(points[:,0], points[:,1], points[:,2], triangles=triangles, cmap=p
lt.cm.Spectral)
    plt.show()
    '''
def kleinbottledemo():
    '''
    Klein bottle simplices and boundary matrices
    '''
    triangles = np.array([[0,1,4],
                          [0,1,7],
                          [0,2,4],
                          [0,2,6],
                          [0,3,6],
                          [0,3,7],
                          [1,2,5],
                          [1,2,8],
                          [1,4,5],
                          [1,7,8],
                          [2,4,8],
                          [2,5,6],
                          [3,4,5],
                          [3,4,8],
                          [3,5,7],
                          [3,6,8],
                          [5,6,7],
                          [6,7,8]
                          ], dtype=int)
    edges = get_subsimplices(triangles)
    points = np.arange(0,9).reshape(-1,1)
    b_matrix2 = boundary_matrix(triangles, edges, format="dok")
    b_matrix1 = boundary_matrix(edges, points, format="dok")
    print "Triangles in K"
    print triangles
    print "Edges in K"
    print edges
    return b_matrix2, b_matrix1
def demo():
    '''
    # Boundary matrices for debugging
    b_matrix2 = np.matrix([[1,0],
                          [1,0],
                          [1,0],
                          [0,1],
                          [0,1],
                          [0,0],
                          [0,1]])
    b_matrix1 = np.matrix([[1,1,0,0,0,0,0],
                          [1,0,1,1,1,0,0],
                          [0,1,1,0,0,1,0],
                          [0,0,0,1,0,1,1],
                          [0,0,0,0,1,0,1]])
    '''
    print "============================================================"
    print "Klein bottle"
    print "============================================================"
    b_matrix2, b_matrix1 = kleinbottledemo()
    global N
    global U
    global V

    start = time.time()
    B = b_matrix2
    N = dok_matrix(B, dtype=int)
    U = dok_matrix(identity(N.shape[0], dtype=int), dtype=int)
    V = dok_matrix(identity(N.shape[1], dtype=int), dtype=int)
    #Smith Normal Form
    print "SNF running:"
    SNF(0, uv=True)
    sparse_savetxt("N2.txt", N, fmt="%d", include_dim=True)
    print "Boundary matrix, B2:\n", B.todense()
    print "SNF2:\n",N.todense()

    # Boundary and cycle ranks
    b2 = 0
    b1 = np.amax(N.nonzero()[1])+1
    z2 = N.shape[1] - b1

    B = b_matrix1
    N = dok_matrix(B, dtype=int)
    U = dok_matrix(identity(N.shape[0], dtype=int), dtype=int)
    V = dok_matrix(identity(N.shape[1], dtype=int), dtype=int)
    print "SNF running:"
    # Smith Normal Form
    SNF(0, uv=True)
    sparse_savetxt("N1.txt", N, fmt="%d", include_dim=True)
    print "Boundary matrix, B1:\n", B.todense()
    print "SNF1:\n",N.todense()

   # Boundary and cycle ranks
    b0 = np.amax(N.nonzero()[1])+1
    z1 = N.shape[1] - b0
    z0 = N.shape[0] #Number of vertices
    print r"Rank of B1: ", b1
    print r"Rank of Z2: ", z2
    print r"Rank of B2: ", b2
    print r"Rank of B0: ", b0
    print r"Rank of Z1: ", z1
    print r"Rank of Z0: ", z0
    print "Betti 0 = (z0-b0): ", z0 - b0
    print "Betti 1 = (z1-b1): ", z1-b1
    print "Betti 2 = (z2-b2):", z2-b2
    print "Betti i, i>2 is trivial since there is no tetrahedra in K"

    '''
    print "U:\n", U.todense()
    print "U inverse:\n", inv(U).todense()
    print "V:\n",V.todense()
    print "U*B*V:"
    tmp = (U*dok_matrix(B, dtype=int)).asformat("dok").astype(int)
    for key, val in tmp.items():
        tmp[key[0], key[1]] = val%2
    tmp = (tmp*V).asformat("dok").astype(int)
    for key, val in tmp.items():
        tmp[key[0], key[1]] = val%2
    print tmp.todense()
    '''

    elapsed = time.time() - start
    print "Time(min) spent: %f" % (elapsed/60)

if __name__ == "__main__":
    demo()
    spinedemo();
```

```
==========================================================
Klein bottle
==========================================================
Triangles in K
[[0 1 4]
 [0 1 7]
 [0 2 4]
 [0 2 6]
 [0 3 6]
 [0 3 7]
 [1 2 5]
 [1 2 8]
 [1 4 5]
 [1 7 8]
 [2 4 8]
 [2 5 6]
 [3 4 5]
 [3 4 8]
 [3 5 7]
 [3 6 8]
 [5 6 7]
 [6 7 8]]
Edges in K
[[1 2]
 [5 6]
 [2 6]
 [4 8]
 [4 5]
 [2 8]
 [1 7]
 [1 4]
 [0 7]
 [3 7]
 [2 5]
 [0 3]
 [3 5]
 [0 1]
 [6 7]
 [6 8]
 [0 6]
 [3 4]
 [5 7]
 [0 2]
 [3 8]
 [1 5]
 [1 8]
 [3 6]
 [0 4]
 [7 8]
 [2 4]]
SNF running:
Boundary matrix, B2:
[[0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0]
 [0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0]
 [1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1]
 [0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0]
 [0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0]
 [0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1]
 [0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]]
SNF2:
[[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
SNF running:
Boundary matrix, B1:
[[0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0]
 [1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0]
 [1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0]
 [0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1]
 [0 1 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0]
 [0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0]
 [0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0]]
SNF1:
[[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
Rank of B1:  17
Rank of Z2:  1
Rank of B2:  0
Rank of B0:  8
Rank of Z1:  19
Rank of Z0:  9
Betti 0 = (z0−b0):  1
Betti 1 = (z1−b1):  2
Betti 2 = (z2−b2): 1
Betti i, i>2 is trivial since there is no tetrahedra in K
```

```
Time(min) spent: 0.001696
==========================================================
Spine
==========================================================
Boundary matrix dimension(EDGESxTRIANGLES):5784x3856
Boundary matrix dimension(VERTICESxEDGES):1928x5784
Rank of B2:  0
Rank of Z2:  1
Rank of B1:  3855
Rank of Z1:  3857
Rank of B0:  1927
Rank of Z0:  1928
Betti 0 = (z0−b0):  1
Betti 1 = (z1−b1):  2
Betti 2 = (z2−b2): 1
Betti i, i>2 is trivial since there is no tetrahedra in K
Time(min) spent: 0.065640
```