dcard@umich.edu

1 Equation Classification

The differential equation (note, subscripts indicate differentiation)

$$aw_{xx} + bw_{xy} + cw_{yy} = f$$

is defined to be hyperbolic/parabolic/elliptic is $b^2 - 4ac$ is positive/zero/negative, respectively. Show that this definition is consistent with the definition in the notes. *Hint: convert the PDE to a system of two equations by defining $u = w_x$ and $v = w_y$*

Firstly noting that $u = w_x$ and that $v = w_y$, this can be written as a system of two equations

$$au_x + bu_y + cv_y = f$$
$$v_x - u_y = 0$$

Defing the state vector as $\vec{u} = \begin{bmatrix} u, & v \end{bmatrix}$, we can write the system in matrix form,

$$\underbrace{\begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix}}_{\underline{\underline{A}}_1} \frac{\partial}{\partial x} \begin{bmatrix} u \\ v \end{bmatrix} + \underbrace{\begin{bmatrix} b & c \\ -1 & 0 \end{bmatrix}}_{\underline{\underline{A}}_2} \frac{\partial}{\partial y} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}$$

Then, since there is no time derivative in this system, classifying the equation for vectors $\vec{k}$ that makes the determinant of $\tilde{A}$ zero or,

$$\det\left(\tilde{A}\right) = \det\left(\underline{\underline{A}}_1 k_1 + \underline{\underline{A}}_2 k_2\right) = \det\left(\begin{bmatrix} ak_1 + bk_2 & ck_2 \\ -k_2 & k_1 \end{bmatrix}\right) = 0$$

Taking the determinant of the 2×2 matrix, we have

$$\left(ak_1 + bk_2\right) k_1 + ck_2^2 = 0$$
$$ak_1^2 + bk_2k_1 + ck_2^2 = 0$$

Dividing through both sides by k_1^2 to get the relation that takes the form of the quadratic expression gives,

$$c\left(\frac{k_2}{k_1}\right)^2 + b\left(\frac{k_2}{k_1}\right) + a = 0$$

Solving for $\frac{k_2}{k_1}$ gives that the relationship is,

$$\boxed{\frac{k_2}{k_1} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2c}}$$

As shown above, this is consistent with the notes since the relation $\mathbf{b^2 - 4ac}$ denotes whether the particular PDE is parabolic/hyperbolic/elliptic depending on whether the roots of $\frac{\mathbf{k_2}}{\mathbf{k_1}}$ are real, have imaginary parts, or are a double root.

2 Gram-Schmidt Orthonormalization and Projection

Given the following three vectors,

$$\vec{u}_1 = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \vec{u}_2 = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 3 \\ 0 \end{bmatrix}, \vec{u}_3 = \begin{bmatrix} 3 \\ 0 \\ 1 \\ 0 \\ -2 \end{bmatrix}$$

a. Use the Gram-Schmidt algorithm to compute an orthonormal basis (three vectors $\vec{v}_1, \vec{v}_2, \vec{v}_3$) for the space spanned by $\{\vec{u}_1, \vec{u}_2, \vec{u}_3\}$. You can write a program to do this.

The Gram-Schmidt algorithm can be defined as,

$$v_n = v_n - \sum_{i=1}^{n-1} \frac{< v_n, v_i >}{< v_i, v_i >} v_i$$

Where $< \phi, \psi >$ is defined as the inner product with the corresponding set respectively such that,

$$< \phi, \psi > = \int_a^b \phi(x)\psi(x)\ dx$$

However, since this is dealing with vectors then applying the discrete version of the Gram-Schmidt algorithm gives,

$$< \vec{\phi}, \vec{\psi} > = \vec{\phi} \cdot \vec{\psi}$$

Applying this algorithm in Python (attached at end of assignment) gives that the three vectors are

$$\boxed{\vec{v}_1 = \begin{bmatrix} 0.44721 \\ 0.89443 \\ 0.00000 \\ 0.00000 \\ 0.00000 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} 0.13188 \\ -0.06594 \\ 0.00000 \\ 0.98907 \\ 0.00000 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} 0.67653 \\ -0.33827 \\ 0.28815 \\ -0.11276 \\ -0.57631 \end{bmatrix}}$$

b. Project the vector $w = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}^T$ into the space spanned by this basis, and write $w = w_{\parallel} + w_{\perp}$, where $w_{\parallel}$ is the projection of w in to the space. Give both $w_{\parallel}$ and $w_{\perp}$.

Projecting the vector onto the space spanned by the basis can be done through the following,

$$\vec{w}_{\parallel,i} = \underbrace{(\vec{w} \cdot \hat{v}_i)}_{\text{Scalar}} \underbrace{\hat{v}_i}_{\text{Direction}}$$

Conducting this over the size of the basis will give the total projection or,

$$\vec{w}_{\parallel} = \sum_{i=1}^{N} (\vec{w} \cdot \hat{v}_i)\, \hat{v}_i$$

Then with the parallel component being defined the perpendicular component is,

$$\vec{w}_{\perp} = \vec{w} - \vec{w}_{\parallel}$$

Using the results from part a.) and looping through the vectors and outputting through Python gives the results to be,

$$\boxed{\vec{w} = \underbrace{\begin{bmatrix} -0.14801 \\ 2.57401 \\ -0.71119 \\ 4.19134 \\ 1.42238 \end{bmatrix}}_{\vec{w}_{\parallel}} + \underbrace{\begin{bmatrix} 1.14801 \\ -0.57401 \\ 3.71119 \\ -0.19134 \\ 3.57762 \end{bmatrix}}_{\vec{w}_{\perp}}}$$

3 Newton-Raphson

Write a computer program that solves the following system of nonlinear equations,

$$x^2 + \sin(y) = 3$$
$$y^3 - \frac{2x}{y} = -10$$

using the Newton-Raphson method, starting with the guess $(x, y) = (1.5, 0.5)$. In your writeup, show all analytical derivations, the final answer, and a convergence history (table form) of the residuals.

The Newton-Raphson method is described by,

$$\Delta U_k = -\left(\frac{\partial R}{\partial U}|_{U_k}\right)^{-1} R(U_k)$$

Where $\frac{\partial R}{\partial U}|_{U_k}$ is the Jacobian matrix of the residual which can be expressed as,

$$\frac{\partial R}{\partial U}|_{U_k} = \begin{bmatrix} \frac{\partial}{\partial x}\left(x^2 + \sin(y) - 3\right) & \frac{\partial}{\partial y}\left(x^2 + \sin(y) - 3\right) \\ \frac{\partial}{\partial x}\left(y^3 - \frac{2x}{y} + 10\right) & \frac{\partial}{\partial y}\left(y^3 - \frac{2x}{y} + 10\right) \end{bmatrix}$$

$$\frac{\partial R}{\partial U}|_{U_k} = \begin{bmatrix} 2x & \cos(y) \\ -\frac{2}{y} & 3y^2 + \frac{2x}{y^2} \end{bmatrix}$$

Then the next iteration can be written to be where ΔU_k is from the definition above,

$$U_{k+1} = U_k + \Delta U_k$$

Implementing this method into Python gives the following results,

Using the method shown above I found that the final value through Newton-Raphson was $(\mathbf{1.63692}, \mathbf{0.32625})$. This was found through 8 iterations but converged to machine precision on iteration 6 as shown in Table 1 below.

Table 1: Python output from Newton-Raphson Method.

Iteration	$(x,\ y)$	$\lvert\vec{e}_k\rvert$
0.0	(1.50000000, 0.50000000)	4.13386e+00
1.0	(1.66929589, 0.22958303)	4.52990e+00
2.0	(1.64522275, 0.29761827)	1.02956e+00
3.0	(1.63762332, 0.32377164)	8.19753e-02
4.0	(1.63692486, 0.32623250)	6.06626e-04
5.0	(1.63691966, 0.32625099)	3.37132e-08
6.0	(1.63691966, 0.32625099)	0.00000e+00
7.0	(1.63691966, 0.32625099)	0.00000e+00
8.0	(1.63691966, 0.32625099)	0.00000e+00

4 Edge Connectivity

In this problem you will write a code that identifies loops of connected points given a scrambled list of edges. The assignment came with two files: `V.txt`, which contains the (x, y) coordinates of the points(nodes); and `E.txt` which contains the list of edges. Each line in `E.txt` corresponds to one edge. It contains two integers, which are the node numbers (numbering starts at 1) of the two endpoints of that edge. The coordinates of node n are given on line n of `V.txt`. A *loop* is an ordered sequence of nodes in which the first and last nodes are the same, and each pair of consecutive nodes is connected by an edge. For a given loop, all of the provided edges have the node pairs numbered consistently(either clockwise or counterclockwise). Write a code that reads in the two files and performs the following:

- Prints out the number of loops
- Starting from the shortest loop and progressing to the longest, prints out the number of unique nodes and the list of nodes in each loop. Start the list of nodes with the smallest-index node number. Format the printing to show 10 numbers per line, with the numbers right-justified in aligned columns.
- Makes a plot of the loops by connecting the ordered nodes in each loop. Use a different color and symbol for each loop. You will find that one of the loops is much further away from the others, so make two plots, one with all of the loops, and one zoomed in to show all but the faraway loop.

In your writeup, explain your approach and algorithm, and include all of the requested output and plots. The documented code should go in the `.zip` file, but we should not have to run it to see the results.

Edge Connectivity Algorithm

1. My algorithm in an overview has two nested while loops with `if` conditional statements to determine the edge connectivity within the loops.
   - First `while` loop is used to loop through all the loops until the algorithm determines that it has found all the interior loops
   - The seconds `while` loop will loop through `E.txt` to find which nodes belongs to the loop in question
2. Then while in these `while` loops, inside I have two `if` statements
   - One `if` statement is used to determine the next node that is connected within the interior loop
   - The other `if` statement is used to determine if it has looped through all the interior nodes and has reached back around to the beginning of the interior loop
3. Outside of the inner `while` loop is another if statement that determines if there are values that still need to be looped over and if so it will pre-allocate values for when it loops through the next interior loop
4. Then once finished with these loops printing the values is done through `unique` and `sort` and remove all 0 entries to remove the possibility of concatenation errors

Implementing my above method, being more involved and extra variables, I was able to `fprintf` to Matlab in the right-justified format with 10 numbers per line. However, the next page I output the values to Table 2 as well. Code will be attached at the end of my assignment.

Table 2: Edge connectivity unique nodes per loop in ascending order.

Loop Number: 1, with 18 unique nodes									
10	196	334	81	112	311	174	15	309	265
265	22	338	254	35	85	97			
Loop Number: 2, with 60 unique nodes									
1	263	222	181	104	169	241	137	187	252
252	70	21	321	210	66	253	38	24	136
136	226	80	274	74	14	307	40	212	277
277	337	87	202	330	213	293	129	162	316
316	336	296	292	320	64	243	92	325	255
255	52	8	329	322	121	99	76	234	
Loop Number: 3, with 122 unique nodes									
2	51	232	256	100	182	122	54	48	89
89	20	185	341	201	247	283	132	180	323
323	4	285	141	7	98	120	82	110	264
264	75	61	128	207	305	229	295	50	271
271	90	91	224	6	95	123	154	157	331
331	279	227	191	291	133	45	248	171	239
239	34	33	262	143	27	233	178	153	39
39	147	200	236	340	209	125	3	314	230
230	145	275	282	335	56	299	151	257	188
188	103	260	149	101	46	30	73	326	290
290	119	206	28	278	111	221	304	18	5
5	317	78	198	166	118	84	208	175	245
245									
Loop Number: 4, with 141 unique nodes									
9	155	114	268	108	267	93	23	144	216
216	127	94	115	197	158	156	116	107	250
250	11	272	168	124	269	319	62	225	161
161	177	96	79	261	184	297	72	134	240
240	289	276	244	266	303	190	298	218	228
228	242	315	86	58	251	126	339	286	43
43	36	288	17	117	172	170	237	219	55
55	146	60	69	71	109	159	42	131	83
83	63	258	204	302	214	29	273	179	53
53	25	176	102	138	194	183	193	327	113
113	259	199	49	12	306	332	173	231	26
26	215	235	246	105	67	211	308	16	189
189	300	249	270	13	313	106	148	238	205
205	65	47	294	160	324	186	152	220	57

Above in Table 2 are the nodal values to the edge values for each given loop. Attached to the end is the tabulated output from Matlab Command Window.

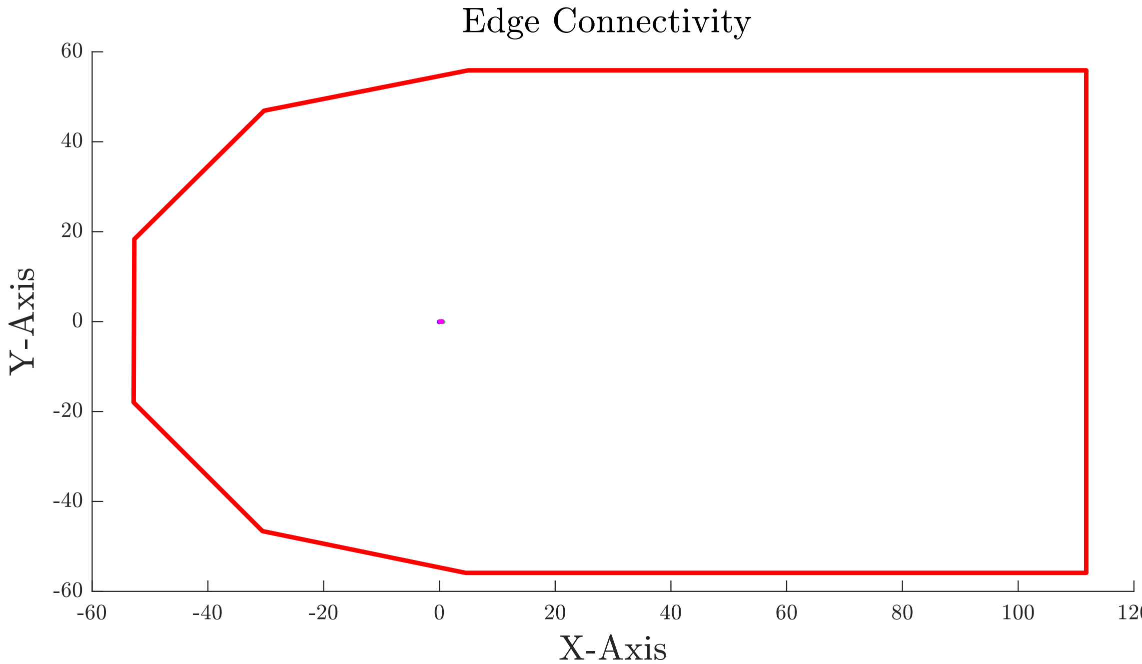

Figure 1: Full view of all loops.

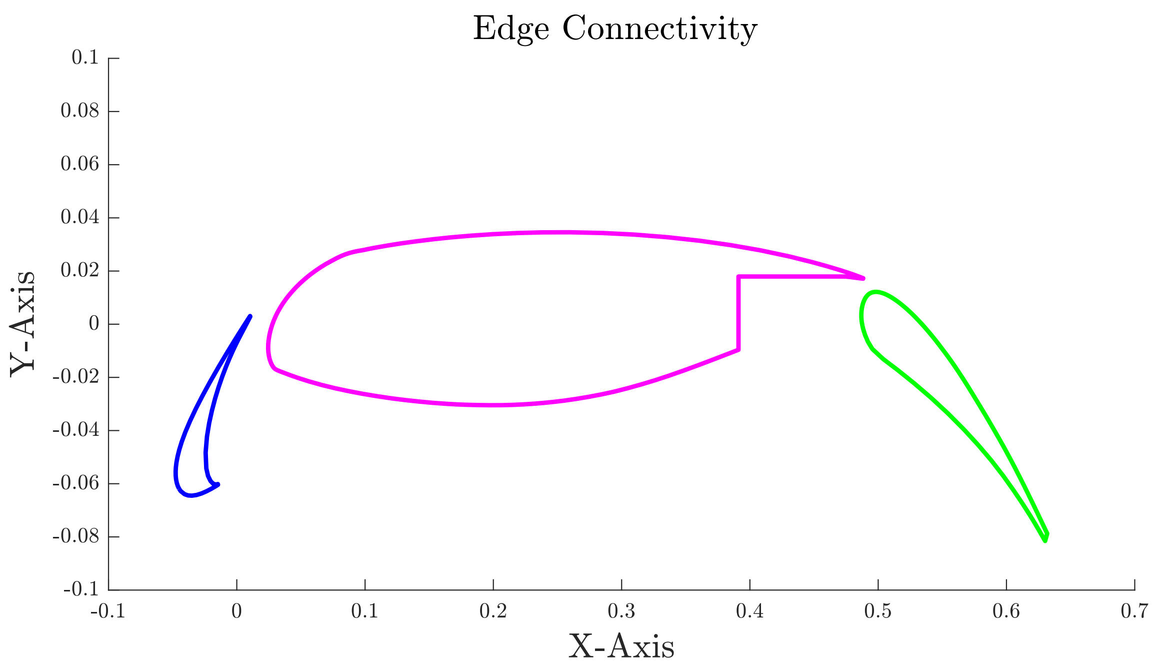

Figure 2: Zoomed in view of all loops.

Looking above to Figures 1,2 I can see that my Matlab code correctly implemented the iterations to determine the correct loops and then succesfully tabulated them to the loops.

Python Code for Gram-Schmidt Orthonormalization

Algorithm 1: Gram-Schmidt Orthonormalization algorithm implementation in Python environment.

```python
import numpy as np
import math

def write_file(v, filename):
    f = open(filename,"w")

    output = ''
    for i in range(len(v)):
        output += ' ' + str.format('{0:.5f}', v[i]) + ' ' + r'\\' # Output results to LaTeX environment
    f.write(output)
    f.close()

def write_w(filename, projection):
    f = open(filename, "w")

    output = ''
    for i in range(len(projection)):
        output += str.format('{0:.5f}', projection[i]) + r'\\ ' # Output results to LaTeX environment
    f.write(output)
    f.close()

# Pre-allocate matrices
u = np.array([[1, 0, 3], [2, -1, 0], [0, 0, 1], [0, 3, 0], [0, 0, -2]])
v = np.zeros(u.shape)

# Apply Gram-Schmidt Algorithm
for i in range(min(u.shape)):
    v[:,i] = u[:,i]
    for j in range(i):
        if j >= 0:
            v[:,i] = v[:,i] - np.dot(v[:,j], v[:,i])*v[:,j] # Apply inner product
    v[:,i] = v[:,i]/math.sqrt(np.dot(v[:,i], v[:,i])) # Normalize

# Output results to LaTeX
write_file(v[:,0], "v1")
write_file(v[:,1], "v2")
write_file(v[:,2], "v3")

w = np.linspace(1, 5, len(v), dtype = int, endpoint=True)
w_parallel = np.zeros(5)
for i in range(min(u.shape)):
    print(i)
    v_norm = v[:,i] / np.linalg.norm(v[:,i])
    w_parallel += np.dot(w,v_norm)*v_norm

w_perp = w - w_parallel
write_w("w_parallel", w_parallel)
write_w("w_perp", w_perp)
```

Python Code for Newton-Raphson Method

Algorithm 2: Newton-Raphson algorithm implementation in Python environment.

```python
import numpy as np
import math

# Pre-allocating values
N = 10 # Number of iterations
u0 = np.matrix([1.5, 0.5]) # Initial guess
u = np.zeros([N, 2])
resid = np.zeros([N,1])
u[0,:] = u0

def f(dat):
    f1 = float(dat[0]**2 + math.sin(dat[1]) - 3) # f_1 function
    f2 = float(dat[1]**3 - 2*dat[0]/dat[1] + 10) # f_2 function

    return np.matrix([[f1], [f2]])

def print_results(Uk, R):
    f = open('q3_results',"w") # Filename

    output = ''
    for i in range(len(u)-1):
        output += str.format('{0:.1f}',i) + r'& ('+str.format('{0:.8f}',Uk[i,0])+', '+str.format('{0:.8f}',Uk[i,1])+ r') &'+str.format('{0:.5e}',R[i,0])+ r'\\' # Output results to LaTeX environment
    f.write(output)
    f.close()

def final_vals(u):
    f = open('q3_final_vals',"w") # Filename

    idx = len(u) - 1
    f.write('(' + str.format('{0:.5f}',u[idx,0])+', '+str.format('{0:.5f}',u[idx,1]) + ')')
    f.close()

resid[0] = float(np.linalg.norm(np.transpose(f(u[0,:]))))
for i in range(N-1):
    jacobian = np.matrix([[2*u[i,0] , math.cos(u[i,1])], [-2/u[i,1], 3*u[i,1]**2 + 2*u[i,0]/(u[i,1]**2)]]) # Compute partial R/partial U @(U_k)
    deltaux = -np.linalg.inv(jacobian) * f(u[i,:]) # Compute the delta_Ux

    u[i+1,:] = u[i,:] + np.array([deltaux[0,0], deltaux[1,0]]) # Compute the next step
    resid[i+1] = np.linalg.norm(np.transpose(f(u[i+1,:]))) # Compute residual

print(resid)
print_results(u, resid) # Output results to LaTeX table
final_vals(u)
```

Matlab Code for Edge Connectivity

Algorithm 3: Edge Connectivity algorithm implementation in Matlab environment.

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
clear all; clc; close all
set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% Load in values for E, V
E = dlmread('E.txt'); Edat = E;
V = dlmread('V.txt');

% Initiate the first loop
loops(1,:) = E(1,:);
E(1,:) = [];
loop_index = [1,2];
while norm(size(E)) > 2.5 % When deleting values norm([1,2]) = 2.236
    k = 2; % Iterator for loop matrix
    i = 1; % Iterator for iterating through E
    inloop = true; % Boolean conditional
    while inloop
        if i == max(size(E)) + 1
            i = 1; % Restart the loop if ran through all the points
        end
        node2 = E(i,2); % The next node in question
        if loops(k-1, loop_index(1)) == node2 % If they are connected
            loops(k, loop_index) = E(i,:); % Set values
            E(i,:) = [];   % Delete entries
            if loops(1, loop_index(2)) == loops(k, loop_index(1))
                inloop = false; % If loop has been completed, break
            elseif size(E)*[1;0] == 1 % Checks for the length of E
                loops(k+1, loop_index) = E(1,:); % Handles the last index
                inloop = false; % Removes itself from the
            end
            k = k + 1;
        else
            i = i + 1;
        end
    end
    if norm(size(E)) > 2.5 % When deleting values norm([1,2]) = 2.236
        loop_index = loop_index + 2; % Increase the loop index by 2
        loops(1:end, loop_index) = zeros(max(size(loops)), 2); % Pre-allocate
        loops(1,loop_index) = E(1,:); % Iniate
        E(1,:) = [];   % Delete entries
    end
end

num_loops = size(loops)*[0;1]/2; % Determine number of loops
fprintf(['The number of loops is ', num2str(num_loops)])
loop_index = [7,8];
for i = 1:num_loops
    dat = loops(1:end, loop_index(1)); % Grab values
    dat(dat == 0) = []; % Delete zero entry

    idx = find(dat == min(dat), 1);

    % Print tabulated values
    fprintf(['\n\nLoop Number: ',num2str(i), ', with ', num2str(max(size(dat))), ' unique nodes\n
        '])
    val_loop = true;
    k = 1;
    looptot = 1;
    while val_loop
        if idx == max(size(dat))
            idx = 1;
        end
        fprintf('%10d ', dat(idx))
        k = k + 1;
        idx = idx + 1;
        looptot = looptot + 1;
        if k == 11
           fprintf('\n')
           k = 1;
```

```
        end
        if looptot == max(size(dat))
           val_loop = false ;
        end
    end
    loop_index = loop_index - 2;
end
%write_to_latex(loops)

loop_index = [7,8];
ax = axes;
ax.ColorOrder = [1 0 0; 0 0 1; 0 1 0; 1 0 1];
ax.LineStyleOrder = {'-','--', '-.', '.'};
hold on
for i = 1:num_loops
    indices1 = loops(1:end, loop_index(1)); indices1(indices1 == 0) = [];
    indices2 = loops(1:end, loop_index(2)); indices2(indices2 == 0) = [];

    node1 = V(indices1,:);
    sz = max(size(node1));
    node1((sz+1):(sz+2),:) = [V(indices2(1),:); V(indices1(1),:)];

    plot(node1(1:end, 1), node1(1:end, 2), 'linewidth', 2)
    loop_index = loop_index - 2;
end
xlabel('X-Axis', 'fontsize', 16)
ylabel('Y-Axis', 'fontsize', 16)
title('Edge Connectivity ', 'fontsize', 16)
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
%export_fig('big_loops.eps')
xlim([-0.1, 0.7])
ylim([-0.1, 0.1])
%export_fig('small_loops.eps')

function write_to_latex(loops)
    fid = fopen('loop_results', 'w');
    loop_index = [7,8];
    num_loops = size(loops)*[0;1]/2;

    for i = 1:num_loops
        dat = loops(1:end, loop_index(1)); % Grab values
        dat(dat == 0) = []; % Delete zero entry
        idx = find(dat == min(dat), 1);

        string = append("Loop Number: ", num2str(i), ', with ', num2str(max(size(dat))), " unique nodes & & & & & & & & & \\ \hline");
        fprintf(fid, '\n %s \n', string);
        val_loop = true;
        k = 1;
        looptot = 1;
        while val_loop
            if idx == max(size(dat))
                idx = 1;
            end
            if k == 10
                lbreak = "\\";
                fprintf(fid, '%10d %s', dat(idx), lbreak);
                fprintf(fid, '\n');
                k = 1;
            else
                fprintf(fid, '%10d &', dat(idx));
                k = k + 1;
                idx = idx + 1;
            end
            looptot = looptot + 1;
            if looptot == max(size(dat))
               val_loop = false ;
            end
        end
        loop_index = loop_index - 2;
    end
end
```

Matlab Command Window Output

```
Command Window
The number of loops is 4

Loop Number: 1, with 18 unique nodes
    10   196   334    81   112   311   174    15   309   265
    22   338   254    35    85    97   167

Loop Number: 2, with 60 unique nodes
     1   263   222   181   104   169   241   137   187   252
    70    21   321   210    66   253    38    24   136   226
    80   274    74    14   307    40   212   277   337    87
   202   330   213   293   129   162   316   336   296   292
   320    64   243    92   325   255    52     8   329   322
   121    99    76   234    37    77   192   150   312

Loop Number: 3, with 122 unique nodes
     2    51   232   256   100   182   122    54    48    89
    20   185   341   201   247   283   132   180   323     4
   285   141     7    98   120    82   110   264    75    61
   128   207   305   229   295    50   271    90    91   224
     6    95   123   154   157   331   279   227   191   291
   133    45   248   171   239    34    33   262   143    27
   233   178   153    39   147   200   236   340   209   125
     3   314   230   145   275   282   335    56   299   151
   257   188   103   260   149   101    46    30    73   326
   290   119   206    28   278   111   221   304    18     5
   317    78   198   166   118    84   208   175   245   318
   223    44    59   130   164   163    41   284   310   140
    88

Loop Number: 4, with 141 unique nodes
     9   155   114   268   108   267    93    23   144   216
   127    94   115   197   158   156   116   107   250    11
   272   168   124   269   319    62   225   161   177    96
    79   261   184   297    72   134   240   289   276   244
   266   303   190   298   218   228   242   315    86    58
   251   126   339   286    43    36   288    17   117   172
   170   237   219    55   146    60    69    71   109   159
    42   131    83    63   258   204   302   214    29   273
   179    53    25   176   102   138   194   183   193   327
   113   259   199    49    12   306   332   173   231    26
   215   235   246   105    67   211   308    16   189   300
   249   270    13   313   106   148   238   205    65    47
   294   160   324   186   152   220    57   203   281    19
    32    31    68   333   217   328   280   301   142   139
fx >>
```

Figure 3: Matlab Command Window output for tabulated node values.