ISSN:2320-0790

An International Journal of Advanced Computer Technology

MATHGRAPH : A PYTHON PACKAGE TO COMPUTE ENERGY AND TOPOLOGICAL INDICES OF GRAPHS

M. R. Rajesh Kanna*
Department of Mathematics,
Sri D Devaraj Urs Government First Grade College,
Hunsur - 571105, India.

R. Pradeep Kumar
Department of Mathematics,
GSSS Institute of Engineering and Technology for Women,
KRS Road, Mysuru - 570 016, India.

Sudharsan Vijayaraghavan
CISCO INC, Software Engineer, Bengaluru, India.

Ananda Kumar M.R
Juniper Networks, Software Engineer, Bengaluru, India.

Soner Nandappa D
Department of Studies in Mathematics, University of Mysore,
Mysuru - 570 006, India.

In this paper, we introduce MathGraph, an open-source and cross-platform Python package. As a Python package, MathGraph is easily integrable with graph visualization softwares. This helps researchers in graph theory to either create a graph programmatically using Python program or draw the graph using Graphical User Interface (GUI) tool such as 'Tulip' to compute distinct sets, energies and topological indices of graphs.

Mathematics Subject Classification : Primary 05C50, 05C15, 05C69.

Keywords and Phrases : MathGraph, Python package, Tulip, Dominating sets, Minimum dominating sets, Minimum dominating energy, Covering sets, Minimum covering sets, Minimum covering energy, Common neighborhood, Laplacian energy, Minimum Laplacian dominating energy, Seidel energy, Maximum degree energy, Atom bond connectivity index, second, fourth and fifth atom bond connectivity index.

I. INTRODUCTION

One of the important applications of graph theory is to represent practical problems by means of structural models. Graph theoretical ideas are highly utilized in computer science applications. Modeling of computer science problems leads to the development of various algorithms. The main objective of this paper is to write a program (MathGraph) in Python language to compute various types of sets, energies and topological indices for any given graph.

MathGraph is made available as a open source python package which can be downloaded from https://pypi.Python.org/pypi/mathgraph. We have used three Python packages namely numpy, networkx and mathchem [6] to implement MathGraph. The MathGraph package can also be used with graph visualization software like i8u8i8iijuTulip. This helps to draw any graph in Tulip using devices like Wacom tablet to find sets, energy and topological indices instantly.

NumPy: NumPy is a open source Python package for scientific computing with Python. NumPy is used in MathGraph to compute eigenvalues and other properties of a graph. Details of this package can be found at

* Department of Mathematics,
Sri D Devaraj Urs Government First Grade College,
Hunsur - 571105, India.; mr.rajeshkanna@gmail.com, pradeepr.mysore@gmail.com, sudvijayr@gmail.com, anandmr.acm@gmail.com and ndsoner@yahoo.co.in

https://pypi.Python.org/pypi/numpy.

NetworkX: NetworkX is a open source Python package for studying graphs and networks. NetworkX is used in MathGraph to create a graph, perform some of the basic operations on a graph such as traversing the edges of the graph, compute neighborhood of the nodes etc., which are helpful in computing sets, energies and topological indices of a graph. Details of this package can be found at https://pypi.Python.org/pypi/networkx.

Mathchem: Mathchem [6] is a open source Python package for calculating topological indices and other invariants of molecular graphs. Mathchem is used in MathGraph to compute degree, order and adjacency matrix of a graph. Details of this package can be found at https://pypi.Python.org/pypi/mathchem.

Tulip: Tulip is an information visualization framework dedicated to the analysis and visualization of relational data. Details of this package can be found at http://tulip.labri.fr.

II. INSTALLATION

Standard Python package installation procedure is good enough for installing mathgraph, mathchem, networkx, numpy and tulip. Typical commands for package installation will be one of the following:

Python setup.py install or pip install <package name> as shown below

```
$pip install numpy
$pip install networkx = 1.10
$pip install mathchem
$pip install tulip
$pip install mathgraph
```

Note: MathGraph is installed and tested on Linux operating system.

III. COMPUTING DISTINCT ENERGIES AND TOPOLOGICAL INDICES

The sets, energies and topological indices of a graph can be computed using MathGraph in two methods:

1. **Standalone method**
2. **Graphical visualization method using Tulip software**

A. Standalone method:

In this method, graph for which sets, energies and topological indices has to be calculated must be created using Python program and compute the values using Python interpreter.

For example:

```
The mathgraph_standalone.py is Python program
and it can be run as shown below

>python mathgraph_standalone.py

Computing covering sets:
Covering sets:
[set([0, 2]), set([0, 1, 2]), set([0, 2, 3]),
set([1, 2, 3]), set([0, 1, 3]),
set([0, 1, 2, 3])]

Minimum covering sets:
[set([0, 2])]

Computing minimum dominating sets:

Computing dominating sets:
Dominating sets:
[set([2]), set([0]), set([0, 1]), set([1, 2]),
set([1, 3]), set([2, 3]), set([0, 3]), set([0, 2]),
set([0, 1, 2]), set([0, 2, 3]), set([1, 2, 3]),
set([0, 1, 3])]

Minimum dominating sets:
[set([2]), set([0])]

Computing common neighborhood:
Common neighborhood:
[[0. 1. 2. 1.]
 [1. 0. 1. 2.]
 [2. 1. 0. 1.]
 [1. 2. 1. 0.]]

Computing energy:
Adjacency matrix:
[[0 1 1 1]
 [1 0 1 0]
 [1 1 0 1]
 [1 0 1 0]]
Eigenvalues are:
[-1.5615528128088303, -1.0000000000000002,
8.881784197000973e-16, 2.561552812808829]
Energy of a graph:
[5.123105625617661]

Computing Laplacian energy:
No of edges:
5
No of vertices:
4
Degree matrix:
[[3. 0. 0. 0.]
 [0. 2. 0. 0.]
 [0. 0. 3. 0.]
 [0. 0. 0. 2.]]
Adjacency matrix:
[[0 1 1 1]
 [1 0 1 0]
 [1 1 0 1]
```

```
 [1 0 1 0]]
Laplacian matrix:
[[ 3. -1. -1. -1.]
 [-1.  2. -1.  0.]
 [-1. -1.  3. -1.]
 [-1.  0. -1.  2.]]
Laplacian eigenvalues:
[1.1102230246251565e-16, 2.0000000000000004,
3.9999999999999996, 4.0]
Laplacian energy of a graph:
[6.0]

Computing seidel energy:
Seidel matrix
[[ 0. -1. -1. -1.]
 [-1.  0. -1.  1.]
 [-1. -1.  0. -1.]
 [-1.  1. -1.  0.]]
Seidel eigenvalues:
[-2.236067977499789, -0.9999999999999999,
1.0, 2.23606797749979]
Seidel energy of a graph:
6.472135954999579

Computing maximum degree energy:
Maximum degree matrix:
[[0. 3. 3. 3.]
 [3. 0. 3. 0.]
 [3. 3. 0. 3.]
 [3. 0. 3. 0.]]
Maximum degree eigenvalues:
[-4.684658438426493, -2.9999999999999996,
8.88178419699941e-16, 7.68465843842649]
Maximum degree energy of a graph:
15.369316876852984

Computing minimum covering energy:

Computing minimum covering sets:

Computing covering sets:
Covering sets:
[set([0, 2]), set([0, 1, 2]), set([0, 2, 3]),
set([1, 2, 3]), set([0, 1, 3]),
set([0, 1, 2, 3])]
Minimum covering sets:
[set([0, 2])]
Minimum covering matrix:
[[1 1 1 1]
 [1 0 1 0]
 [1 1 1 1]
 [1 0 1 0]]
Minimum covering eigenvalues:
[-1.2360679774997896, -4.319753644032946e-17,
7.09531120559584e-17, 3.23606797749979]
Minimum covering energy of a graph:
[4.47213595499958]

Computing minimum dominating energy:

Computing minimum dominating sets:

Computing dominating sets:
Dominating sets:
[set([2]), set([0]), set([0, 1]), set([1, 2]),
set([1, 3]), set([2, 3]), set([0, 3]),
set([0, 2]), set([0, 1, 2]), set([0, 2, 3]),
set([1, 2, 3]), set([0, 1, 3])]
Minimum dominating sets:
[set([2]), set([0])]
Minimum dominating matrix:
[[1 1 1 1]
 [1 0 1 0]
 [1 1 1 1]
 [1 0 1 0]]
Minimum dominating eigenvalues:
[-1.2360679774997896, -4.319753644032946e-17,
7.09531120559584e-17, 3.23606797749979]
Minimum dominating energy of a graph:
[4.47213595499958]

Computing minimum dominating Laplacian energy:

Computing minimum dominating sets:

Computing dominating sets:
Dominating sets:
[set([2]), set([0]), set([0, 1]), set([1, 2]),
set([1, 3]), set([2, 3]), set([0, 3]),
set([0, 2]), set([0, 1, 2]), set([0, 2, 3]),
set([1, 2, 3]),  set([0, 1, 3])]
Minimum dominating sets:
[set([2]), set([0])]
Degree matrix:
[[3. 0. 0. 0.]
 [0. 2. 0. 0.]
 [0. 0. 3. 0.]
 [0. 0. 0. 2.]]
Minimum dominating  matrix:
[[0 1 1 1]
 [1 0 1 0]
 [1 1 1 1]
 [1 0 1 0]]
Minimum dominating Laplacian matrix:
[[ 3. -1. -1. -1.]
 [-1.  2. -1.  0.]
 [-1. -1.  2. -1.]
 [-1.  0. -1.  2.]]
Minimum dominating Laplacian eigenvalues:
[-0.3027756377319952, 2.0, 3.3027756377319952, 4.0]
Minimum dominating Laplacian energy of a graph:
[9.60555127546399]

Computing Atom-Bond Connectivity Index:
Atom-Bond Connectivity Index:
3.4950937914128569206
```

```
Computing Atom-Bond Connectivity Index2:
Atom bond connectivity index2:
0.0

Computing Atom-Bond Connectivity Index4:
Atom bond connectivity index4:
6.716645127367839

Computing Atom-Bond Connectivity Index5:
atom bond connectivity index5:
0.0
```

Below is the code for mathgraph_standalone.py:

```
import mathgraph as mg
#import numpy as np
#from numpy import linalg as la

G = mg.MathGraph()
# for example adding a triangle

G.add_edge(0,1)
G.add_edge(1,2)
G.add_edge(2,3)
G.add_edge(3,0)
G.add_edge(0,2)

G.minimum_covering_set()
G.minimum_dominating_set()
G.common_neighbourhood()
G.energy()
G.laplacian_energy()
G.seidel_energy()
G.maximum_degree_energy()
G.minimum_covering_energy()
G.minimum_dominating_energy()
G.minimum_dominating_Laplacian_energy()
G.atom_bond_connectivity_index()
G.atom_bond_connectivity_index2()
G.atom_bond_connectivity_index4()
G.atom_bond_connectivity_index5()
```

B. Graphical visualization method

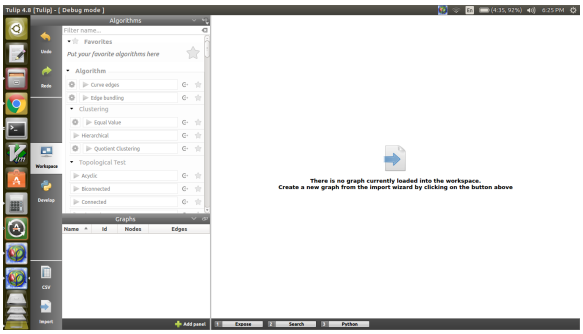
In this method, graph has to be drawn using Tulip software. Compute the values of sets, energies and topological indices using Python plugin script as shown below.

Step 1: To start the Tulip software type tulip in the shell prompt as shown below

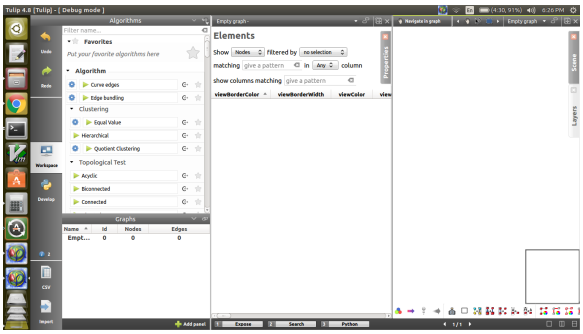
tulip

Step 2: Above command will open tulip application as shown below.

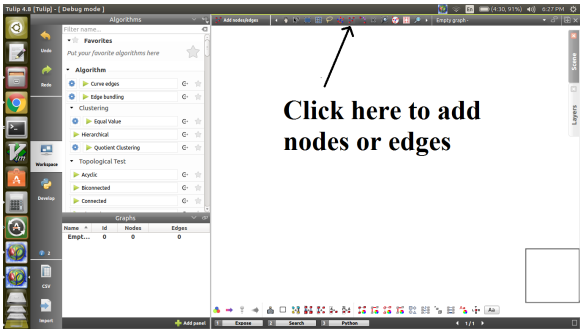
Click on the arrow button to get the graph importing wizard.



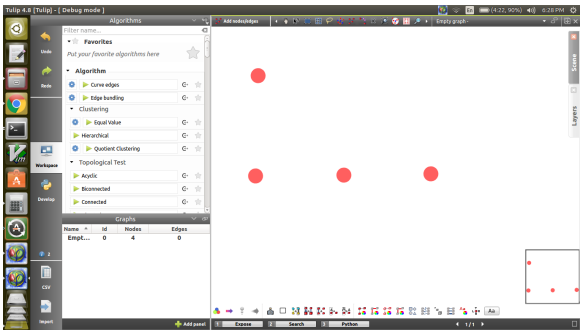
Step 4: Any graph can be drawn in the right most window



Step 5: Click on add nodes/edges icon pointed by the arrow mark in the screen below

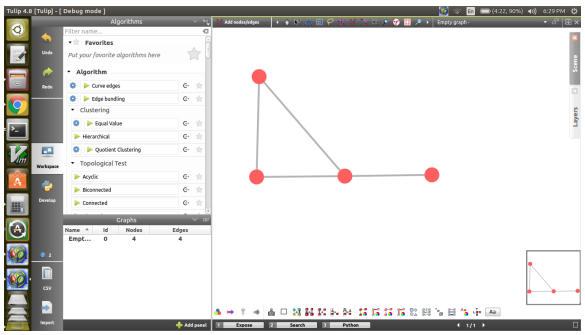


Step 6: Add the number of nodes as required to draw the graph

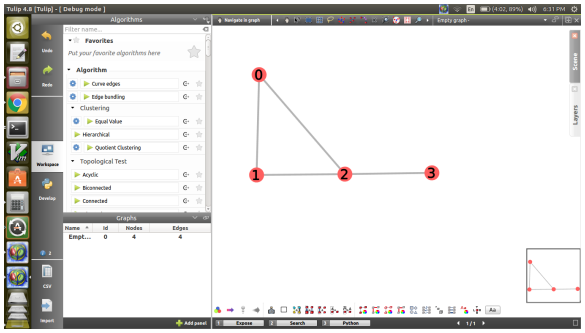


Step 7: Draw the edges by dragging the mouse between

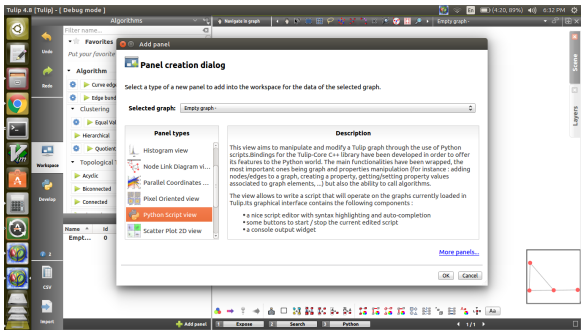
the nodes as shown in the screen below



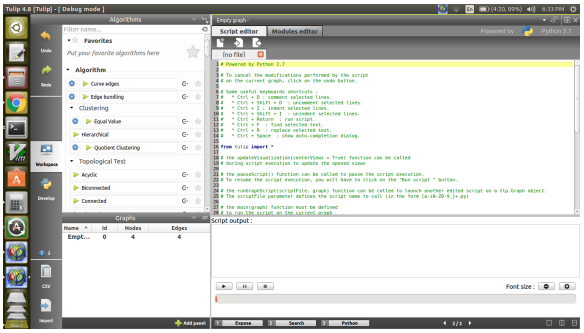
Step 8: If required label the nodes by following steps below:
a) First click on hand symbol as pointed by the arrow mark
b) Select a node to be labeled and then right click to select the edit option
c) Click on label and number the nodes as desired.



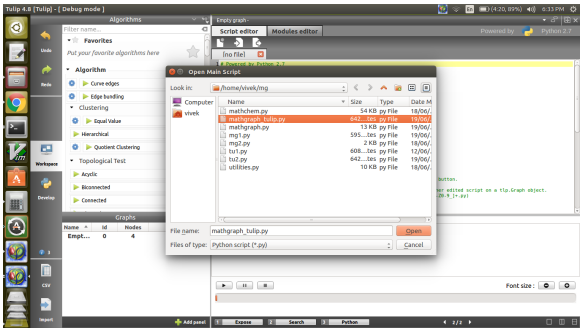
Step 9: click on add panel button to get panel description dialog as shown in the figure



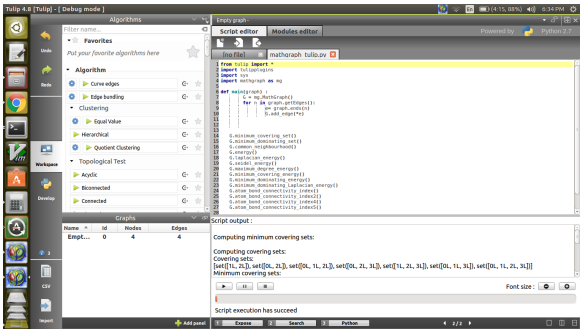
Step 10: Select python script view then click ok



Step 11: In the script editor window, click on load file button and select mathgraph tulip Python file as shown file



Step 12: Finally run the Python script by clicking the play button in the script output window.



The sets, energies and topological indices of the drawn graph will be displayed in the script output window.

IV. TYPES OF ENERGIES

The concept of energy of a graph was introduced by I. Gutman [14] in 1978. Let G be a graph with n vertices and m edges. Let $V(G) = \{v_1, v_2, v_3, \ldots, v_n\}$ and $E(G) = \{e_1, e_2, e_3, \ldots, e_m\}$ be the vertex set and edge set of a graph. The **adjacency matrix** of the graph is $n \times n$ matrix $A = (a_{ij})$ where its elements a_{ij} are defined by

$$a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E(G), \\ 0, & \text{if } (v_i, v_j) \notin E(G). \end{cases}$$

The characteristic equation of G is $|A - \lambda I| = 0$. The roots of this equation $\lambda_1, \lambda_2, \ldots, \lambda_n$ are called characteristic roots or eigenvalues of A (or G), which are usually taken in increasing order. The greatest eigenvalue λ_1 is called spectral radius of G. Here A is a real symmetric matrix with real eigenvalues of G whose sum is zero.

The collection of eigenvalues of adjacency matrix is called the spectrum of G. If $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$ are the distinct eigenvalues of G with multiplication $m_1, m_2, \ldots, m_k$ respectively then $\text{spec}(G) = \begin{pmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_k \\ m_1 & m_2 & \cdots & m_k \end{pmatrix}$

The **energy** $E(G)$ is defined to be the sum of the absolute values of the eigenvalues of G. i.e., $E(G) = \sum_{i=1}^{n} |\lambda_i|$.

For details on the mathematical aspects of the theory of graph energy, basic properties including various upper and lower bounds for energy of a graph can be found in [16].

A. Minimum Covering Energy

In the year 2012, C. Adiga et al. [1] introduced the minimum covering energy of a graph. Let G be a simple graph of order n with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set E. A subset C of V is called a covering set of G if every edge of G is incident to at least one vertex of V. Any covering set with minimum cardinality is called a minimum covering set. Let C be a minimum covering set of a graph G. The minimum covering matrix of G is the $n \times n$ matrix defined by $A^C(G) := (a_{ij})$ where $a_{ij} = \begin{cases} 1, & \text{if } v_i v_j \in E, \\ 1, & \text{if } i = j \text{ and } v_i \in C, \\ 0, & \text{otherwise.} \end{cases}$

The characteristic polynomial of $A^C(G)$ is denoted by $f_n(G, \lambda) = det(\lambda I - A^C(G))$. The minimum covering eigenvalues of the graph G are the eigenvalues of $A^C(G)$. Since $A^C(G)$ is real and symmetric, its eigenvalues are real numbers and we label them in non-increasing order $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_n$. The **minimum covering energy** of G is then defined as $E^C(G) = \sum_{i=1}^{n} |\lambda_i|$.

B. Minimum Dominating Energy

M. R. Rajesh Kanna et al. [17] introduced minimum dominating energy of a graph $E_D(G)$.

Let G be a simple graph of order n with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set E. A subset D of V is called a dominating set of G if every vertex of $V - D$ is adjacent to some vertex in D. Any dominating set with minimum cardinality is called a minimum dominating set. Let D be a minimum dominating set of a graph G. The minimum dominating matrix of G is the $n \times n$ matrix defined by $A_D(G) = (a_{ij})$,

where $a_{ij} = \begin{cases} 1, & \text{if } v_i v_j \in E, \\ 1, & \text{if } i = j \text{ and } v_i \in D, \\ 0, & \text{otherwise.} \end{cases}$

The characteristic polynomial of $A_D(G)$ is denoted by $f_n(G, \lambda) = det(\lambda I - A_D(G))$. The minimum dominating eigenvalues of the graph G are the eigenvalues of $A_D(G)$. Since $A_D(G)$ is real and symmetric, its eigenvalues are real numbers and we label them in non-increasing order $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_n$. The minimum dominating energy of G is defined as $E_{DM}(G) = \sum_{i=1}^{n} |\lambda_i|$.

Note that the trace of $A_D(G)$ = Domination number = k.

C. Laplacian Energy

I. Gutman and B. Zhou [15] introduced the Laplacian energy of a graph G in the year 2006.

Let G be a graph with n vertices and m edges. The **Laplacian matrix** of the graph G, denoted by $L = (L_{ij})$, is a square matrix of order $n \times n$ whose elements are defined as

$$L_{ij} = \begin{cases} -1, & \text{if } v_i \text{ and } v_j \text{ are adjacent,} \\ 0, & \text{if } v_i \text{ and } v_j \text{ are not adjacent,} \\ d_i, & \text{if } i = j \ . \end{cases}$$

where d_i is the degree of the vertex v_i. Let $\mu_1, \mu_2, \ldots, \mu_n$ be the Laplacian eigenvalues of G then **Laplacian energy** $LE(G)$ of G is defined as $LE(G) = \sum_{i=1}^{n} \left| \mu_i - \frac{2m}{n} \right|$.

D. Minimum Laplacian Dominating Energy

M. R. Rajesh Kanna et al. [17] introduced minimum Laplacian dominating energy of a graph $E_D(G)$.

Let $D(G)$ be the diagonal matrix of vertex degrees of the graph G. Then $L_D(G) = D(G) - A_D(G)$ is called the minimum Laplacian dominating matrix of G. Let μ_1, μ_2, μ_3, …, μ_n be the eigenvalues of $L_D(G)$, arranged in non-increasing order. These eigenvalues are called minimum Laplacian dominating eigenvalues of G. The **minimum Laplacian dominating energy** of the graph G is defined as $LE_D(G) = \sum_{i=1}^{n} \left| \mu_i - \frac{2m}{n} \right|$.

where m is the number of edges of G and $\frac{2m}{n}$ is the average degree of G.

E. Seidel Energy

Willem H. Haemers [18] defined Seidel energy of a graph. The Seidel matrix of G is the $n \times n$ matrix denoted by $S(G) = (s_{ij})$, where $s_{ij} = \begin{cases} -1, & \text{if } v_iv_j \in E, \\ 1, & \text{if } v_iv_j \neq E, \\ 0, & \text{if } v_iv_j. \end{cases}$

The characteristic polynomial of $S(G)$ is denoted by $f_n(G, \lambda) = det(\lambda I - S(G))$. The Seidel eigenvalues of the graph G are the eigenvalues of $S(G)$. Since $S(G)$ is real and symmetric, its eigenvalues are real numbers. The **Seidel energy** of G defined as $SE(G) = \Sigma_{i=1}^{n} |\lambda_i|$.

F. Maximum Degree Energy

C. Adiga and M. Smitha [2] defined maximum degree energy of a graph. The maximum degree matrix of G is the $n \times n$ matrix defined by $A_{MD}(G) = (a_{ij})$, where $a_{ij} = \begin{cases} max\{dv_i, dv_j\} & \text{if } v_iv_j \in E \\ 0 & otherwisw \end{cases}$

The characteristic polynomial of $A_{MD}(G)$ is denoted by $f_n(G, \lambda) = det(\lambda I - A_{MD}(G))$. The maximum degree eigenvalues of the graph G are the eigenvalues of $A_{MD}(G)$. Since $A_{MD}(G)$ is real and symmetric, its eigenvalues are real numbers and we label them in non-increasing order $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. The **maximum degree energy** of G defined as $E_{MD}(G) = \sum_{i=1}^{n} |\lambda_i|$.

V. TOPOLOGICAL INDICES

All graphs considered in this thesis are finite, connected, loop less and without multiple edges. Let $G(V, E)$ be a graph with n vertices and m edges. The degree of a vertex $u \in V(G)$ is denoted by d_u and is the number of vertices that are adjacent to u. The edge connecting the vertices u and v is denoted by uv.

Topological indices are the molecular descriptors that describe the structures of chemical compounds and they help us to predict certain physico-chemical properties like boiling point, enthalpy of vaporization, stability, etc. Molecules and molecular compounds are often modeled by molecular graph. A molecular graph is a representation of the structural formula of a chemical compound in terms of Graph theory, whose vertices correspond to the atoms of the compound and edges correspond to chemical bonds. Note that hydrogen atoms are often omitted.

A topological index $Top(G)$ of a graph G, is a number with the property that for every graph H isomorphic to G, $Top(H) = Top(G)$. So, a topological index is a real number derived from the structure of a graph, which is invariant under graph isomorphism. Topological indices are numerical parameters of a graph which characterize its topology and are usually graph invariants.

A graph invariant is any function on a graph that does not depend on a labeling of its vertices. A topological index is a graph invariant applicable in chemistry. By IUPAC terminology, a topological index is a numerical value associated with chemical constitution purporting for correlation of chemical structure with various physical properties, chemical reactivity or biological activity.

A. Atom Bond Connectivity index

The Atom-bond connectivity index, ABC index is one of the degree based molecular descriptor, which was introduced by Estrada et al. [8] in late 1990's and it can be used for modeling thermodynamic properties of organic chemical compounds, it is also used as a tool for explaining the stability of branched alkanes [7]. Some upper bounds for the atom-bond connectivity index of graphs can be found in [4], The atom-bond connectivity index of chemical bicyclic graphs, connected graphs can be seen in [5, 20]. For further results on ABC index of trees see the papers [11, 19, 21] and the references cited there in.

Let $G(V, E)$ be a molecular graph and d_u is the degree of the vertex u, then ABC index of G is defined as, $ABC(G) = \sum_{uv \in E} \sqrt{\frac{d_u + d_v - 2}{d_u d_v}}$.

Recently, Graovac and Ghorbani, introduced a new version of the atom-bond connectivity index namely the second atom bond connectivity index [13].

$ABC_2(G) = \sum_{uv \in E} \sqrt{\frac{n_u + n_v - 2}{n_u n_v}}$ where n_u is the number of vertices closer to vertex u than vertex v and n_v defines similarly.

The fourth atom bond connectivity index, $ABC_4(G)$ index was introduced by M. Ghorbani et al. [12] in 2010. Further studies on $ABC_4(G)$ index can be found in [9, 10].

Let G be a graph, then its fourth ABC index is defined as, $ABC_4(G) = \sum_{uv \in E(G)} \sqrt{\frac{S_u + S_v - 2}{S_u S_v}}$, where S_u is sum of the degrees of all neighbors of vertex u in G. In other words, $S_u = \sum_{uv \in E(G)} d_v$, Similarly for S_v.

The fifth atom bond connectivity index, $ABC_5(G)$ index was introduced by Calimli M.H. [3] in 2011.

Let G be a graph, then its fifth ABC index is defined as, $ABC_5(G) = \sum_{uv \in E(G)} \sqrt{\frac{M_u + M_v - 2}{M_u M_v}}$, where M_u denotes the products of the degrees of adjacent vertices of u. Similarly for M_v.

In this paper, we determine dominating sets, minimum dominating sets, minimum dominating energy, covering sets, minimum covering sets, minimum covering energy, common neighborhood, Laplacian energy, minimum Laplacian dominating energy, Seidel energy, maximum degree energy, atom bond connectivity index, second, fourth and fifth atom bond connectivity index of a graph by using Mathgraph package.

VI. MATHGRAPH PROGRAM

Mathgraph program to compute various types of energies and topological indices for a graph.

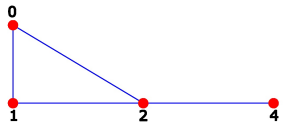
```
import numpy as np
import networkx as nx
import mathchem as mc
import itertools as it
from numpy import linalg as la
class MathGraph ():
    r"""
    MathGraph
    """

    __NX_graph = None
    __Mol_graph = None


    def _reset_(self):
        """ Reset all attributes """
        self.__NX_graph = None
        self.__Mol_graph = None

    def __init__(self):
        """ Molecular graph class """
self.__NX_graph = nx.Graph()

    def NX_graph(self):
        """ Return NetworkX graph object """
        return self.__NX_graph

    def Mol_graph(self):
        """ Return Mathchem graph object """
if self.__Mol_graph is None:
nxg = self.NX_graph()
self.__Mol_graph = mc.Mol(nx.generate_graph6(nxg))
return self.__Mol_graph

    def add_edge(self, u, v=None):
"""Add edge between u and v"""
nxg = self.NX_graph()

if isinstance(u, (int, long)) == False:
x = u.id
else:
x = u
if isinstance(u, (int, long)) == False:
y = v.id
else:
y = v

nxg.add_edge(x, y)

    def degree_matrix(self):
""" degrees matrix """
nxg = self.NX_graph()
mcg = self.Mol_graph()
nodes = nxg.nodes()
   DM = np.ndarray((len(nodes), len(nodes)))
degrees = np.array(mcg.degrees())
for i in nodes:
for j in nodes:
if i == j:
DM[i,i] = degrees[i]
else:
DM[i,j] = 0

return DM

    def subset (self):
        """ find subsets of nodes """
        nxg = self.NX_graph()
nodes = nxg.nodes()
subsets_array = []
num = 1
for i in nodes:
subsets = set(it.combinations(nodes,num))
subsets_array.append([])
for j in subsets:
t = set(list(j))
subsets_array[num-1].append(t)
num += 1
return subsets_array

    def complementary_subset(self):
""" find complement sets for every subset of nodes """
subsets_array = self.subset()
        nxg = self.NX_graph()
nodes = nxg.nodes()
nodes_set = set([])
num = 1
comp_subsets_array = []
for i in nodes:
nodes_set.add(i)
for i in nodes:
comp_subsets_array.append([])
diffs = set([])
for j in subsets_array[num-1]:
diffs = nodes_set - j
comp_subsets_array[num-1].append(diffs)
num += 1
return comp_subsets_array


    def dominating_condition(self, k, num, count):
""" logic to verify presence of dominance """
```

```
        subsets_array = self.subset()
        nxg = self.NX_graph()
nodes = nxg.nodes()
subset_row = subsets_array[num-1]
lcount = 1
for j in subset_row:
if lcount == count:
for m in j:
if nxg.has_edge(k,m):
return 1
lcount += 1

return 0


    def dominating_set(self):
""" dominating sets of a graph """
comp_subsets_array = self.complementary_subset()
nodes_set = set([])
dom_set = []
num = 1
        nxg = self.NX_graph()
nodes = nxg.nodes()
num = 1
for i in nodes:
nodes_set.add(i)
for i in nodes:
comp_subset_row = comp_subsets_array[num-1]
count = 1
for j in comp_subset_row:
ret = 0
for k in j:
ret = self.dominating_condition(k, num, count)
if ret == 0:
break;
if ret == 1:
dom_set.append(nodes_set - j)
count += 1
num += 1
return dom_set

    def minimal_dominating_set(self):
""" minimal dominating sets of a graph """
minimal_dom_set = []
dom_set = self.dominating_set()
dom_len = len(dom_set[0])
for i in dom_set:
if dom_len != len(i):
break;
minimal_dom_set.append(i)
return minimal_dom_set

    def minimal_dominating_energy(self):
""" Minimal dominating energy """
minimal_dom_set = self.minimal_dominating_set()
print minimal_dom_set
energy = []
nxg = self.NX_graph()
mcg = self.Mol_graph()
nodes = nxg.nodes()
adj = np.array(mcg.adjacency_matrix())
for j in minimal_dom_set:
for u, v in nxg.edges_iter():
if u in j and v in j:
adj[u,v] = 1
for i in nodes:
if i in j:
adj[i,i] = 1
s = la.eigvalsh(adj).tolist()
s.sort(reverse=True)
a = np.sum(s,dtype=np.longdouble)/len(s)
energy.append(np.float64(np.sum( map(
lambda x: abs(x-a) ,s), dtype=np.longdouble)))

return energy

    def covering_set(self):
""" covering sets of a graph """
cover_set = []
nxg = self.NX_graph()
nodes = nxg.nodes()
        subsets_array = self.subset()
num = 1
for i in nodes:
subset_row = subsets_array[num-1]
         for j in subset_row:
flag = 0
for u, v in nxg.edges_iter():
if u in j or v in j:
flag = 1
else:
flag = 0
break;
if flag == 1:
cover_set.append(j)
num += 1
return cover_set

    def minimal_covering_set(self):
""" covering sets of a graph """
minimal_cover_set = []
cover_set = self.covering_set()
cover_len = len(cover_set[0])
for i in cover_set:
if cover_len != len(i):
break;
minimal_cover_set.append(i)

return minimal_cover_set

    def minimal_covering_energy(self):
""" Minimal covering energy """
minimal_cover_set = self.minimal_covering_set()
print minimal_cover_set
energy = []
nxg = self.NX_graph()
```

```
mcg = self.Mol_graph()
nodes = nxg.nodes()
adj = np.array(mcg.adjacency_matrix())
for j in minimal_cover_set:
for u, v in nxg.edges_iter():
if u in j and v in j:
adj[u,v] = 1
for i in nodes:
if i in j:
adj[i,i] = 1
s = la.eigvalsh(adj).tolist()
s.sort(reverse=True)
a = np.sum(s,dtype=np.longdouble)/len(s)
energy.append(np.float64(np.sum( map(
lambda x: abs(x-a), s), dtype=np.longdouble)))

return energy

    def min_laplacian_dominating_energy(self):
""" laplacian covering energy """
nxg = self.NX_graph()
minimal_dom_set = self.minimal_dominating_set()
energy = []
nodes = nxg.nodes()
mcg = self.Mol_graph()
degree_array = np.array(self.degree_matrix())
adj = np.array(mcg.adjacency_matrix())
for j in minimal_dom_set:
for u, v in nxg.edges_iter():
if u in j and v in j:
adj[u,v] = 1
for i in nodes:
if i in j:
adj[i,i] = 1
print adj
lap = adj - degree_array
print lap
s = la.eigvalsh((lap)).tolist()
s.sort(reverse=True)
a = np.sum(s,dtype=np.longdouble)/len(s)
energy.append(np.float64(np.sum( map(
lambda x: abs(x-a), s), dtype=np.longdouble)))

return energy

    def atom_bond_connectivity_index2(self):
    """ Atom-Bond Connectivity Index (ABC2) """
nxg = self.NX_graph()
mcg = self.Mol_graph()
        s = np.longdouble(0) # summator
la = mcg.edges()
lb = mcg.vertices()
        for (x,y) in nxg.edges():
         s1 = np.longdouble(0) # summator
         s2 = np.longdouble(0) # summator
t1 = []
t2 = []
la = mcg.distances_from_vertex(x)
lb = mcg.distances_from_vertex(y)
for  keys,values in la.items():
t1.append(values)
for  keys,values in lb.items():
t2.append(values)
for v in mcg.vertices():
if t1[v]<t2[v]:
s1 += 1
elif t1[v]>t2[v]:
s2 += 1

if s1 != 0 and s2 != 0:
s += np.longdouble(
( (s1 + s2 - 2 ) / (s1 * s2)) ** .5 )

        return np.float64(s)

    def atom_bond_connectivity_index4(self):
        """ Atom-Bond Connectivity Index (ABC4) """
        nxg = self.NX_graph()
mcg = self.Mol_graph()
        s = np.longdouble(0) # summator
        for (x,y) in nxg.edges():
        s1 = np.longdouble(0) # summator
        s2 = np.longdouble(0) # summator
        l = nx.all_neighbors(nxg, x)
        m = nx.all_neighbors(nxg, y)
                for i in l:
                    s1 += np.float64(mcg.degrees()[i])
                for i in m:
                    s2 += np.float64(mcg.degrees()[i])
if s1 != 0 and s2 != 0:
                 s += np.longdouble(
                ( (s1 + s2 - 2 ) / (s1 * s2)) ** .5 )
        return np.float64(s)

    def seidel_energy(self):
        """ seidel energy """
mcg = self.Mol_graph()
        s = la.eigvalsh(mcg.seidel_matrix()).tolist()
        s.sort(reverse=True)
        a = np.sum(s,dtype=np.longdouble)/len(s)
        return np.float64(np.sum( map(
 lambda x: abs(x-a) ,s), dtype=np.longdouble))

    def maximum_degree_energy(self):
                """ Max degree energy """
mcg = self.Mol_graph()
                m = mcg.order()
                n = mcg.vertices()
                RD = np.ndarray((m, m))
                for i in n:
                        for j in n:
                 if mcg.distance_matrix()[i,j] == 1:
 RD[i,j] = np.maximum(mcg.degrees()[i], mcg.degrees()[j])
                 else:
                 RD[i,j] = 0;
s = la.eigvalsh(RD).tolist()
```

```
  s.sort(reverse=True)
 a = np.sum(s,dtype=np.longdouble)/len(s)
 return np.float64(np.sum( map(
lambda x: abs(x-a),s), dtype=np.longdouble))

    def common_neighbourhood(self):
                """ common neighborhood """
                nxg = self.NX_graph()
mcg = self.Mol_graph()
                m = mcg.order()
                n = mcg.vertices()
                RD = np.ndarray((m, m))
                for i in n:
                        for j in n:
                        if i == j:
                          RD[i, j] = 0
                          continue
 l =  nx.common_neighbors(nxg, i, j)
                           count = 0
                           for k in l:
                            count += 1
                RD[i,j] = count
                return RD

    def atom_bond_connectivity_index5(self):
 """ Atom-Bond Connectivity Index (ABC5) """
        nxg = self.NX_graph()
        s = np.longdouble(0) # summator
        for (x,y) in nxg.edges():
          ex = np.longdouble(0) # summator
          ey = np.longdouble(0) # summator
          ex = nx.eccentricity(nxg, x)
          ey = nx.eccentricity(nxg, y)
if ex != 0 and ey != 0:
s += np.longdouble(
( (ex + ey - 2 ) / (ex * ey)) ** .5 )
        return np.float64(s)
```

Illustration : 1. Consider a graph,



Screen Output:

```
Computing minimum covering sets:

Computing covering sets:
Covering sets:
[set([1L, 2L]), set([0L, 2L]),
set([0L, 1L, 2L]), set([0L, 2L, 3L]),
set([1L, 2L, 3L]), set([0L, 1L, 3L]),
set([0L, 1L, 2L, 3L])]
Minimum covering sets:
[set([1L, 2L]), set([0L, 2L])]

Computing minimum dominating sets:
Computing dominating sets:
Dominating sets:
[set([2L]), set([1L, 2L]), set([1L, 3L]),
set([2L, 3L]), set([0L, 3L]), set([0L, 2L]),
set([0L, 1L, 2L]), set([0L, 2L, 3L]),
set([1L, 2L, 3L]), set([0L, 1L, 3L])]
Minimum dominating sets:
[set([2L])]

Computing common neighborhood:
Common neighborhood:
[[0. 1. 1. 1.]
 [1. 0. 1. 1.]
 [1. 1. 0. 0.]
 [1. 1. 0. 0.]]

Computing energy:
Adjacency matrix:
[[0 1 1 0]
 [1 0 1 0]
 [1 1 0 1]
 [0 0 1 0]]
Eigenvalues are:
[-1.4811943040920155, -1.0,
0.31110781746598215, 2.1700864866260337]
Energy of a graph:
[4.962388608184031]

Computing Laplacian energy:
No of edges:
4
No of vertices:
4
Degree matrix:
[[2. 0. 0. 0.]
 [0. 2. 0. 0.]
 [0. 0. 3. 0.]
 [0. 0. 0. 1.]]
Adjacency matrix:
[[0 1 1 0]
 [1 0 1 0]
 [1 1 0 1]
 [0 0 1 0]]
Laplacian matrix:
[[ 2. -1. -1.  0.]
 [-1.  2. -1.  0.]
 [-1. -1.  3. -1.]
 [ 0.  0. -1.  1.]]
Laplacian eigenvalues:
[-1.7752834956276696e-16,
0.9999999999999999, 3.0,
 4.000000000000002]
Laplacian energy of a graph:
[6.000000000000002]

Computing seidel energy:
Seidel matrix
[[ 0. -1. -1.  1.]
```

```
 [-1.  0. -1.  1.]
 [-1. -1.  0. -1.]
 [ 1.  1. -1.  0.]]
Seidel eigenvalues:
[-2.2360679774997885, -1.0,
1.0000000000000002, 2.23606797749979]
Seidel energy of a graph:
6.472135954999579

Computing maximum degree energy:
Maximum degree matrix:
[[0. 2. 3. 0.]
 [2. 0. 3. 0.]
 [3. 3. 0. 3.]
 [0. 0. 3. 0.]]
Maximum degree eigenvalues:
[-4.645751311064589, -2.0000000000000004,
0.6457513110645907,
 6.000000000000003]
Maximum degree energy of a graph:
13.291502622129183

Computing minimum covering energy:

Computing minimum covering sets:

Computing covering sets:
Covering sets:
[set([1L, 2L]), set([0L, 2L]), set([0L, 1L, 2L]),
set([0L, 2L, 3L]), set([1L, 2L, 3L]),
set([0L, 1L, 3L]), set([0L, 1L, 2L, 3L])]
Minimum covering sets:
[set([1L, 2L]), set([0L, 2L])]
Minimum covering matrix:
[[1 1 1 0]
 [1 1 1 0]
 [1 1 1 1]
 [0 0 1 0]]
Minimum covering eigenvalues:
[-0.8608058531117033, -6.76685531271428e-16,
0.7458983116349476,
3.114907541476756]
Minimum covering energy of a graph:
[4.721611706223408]

Computing minimum dominating energy:

Computing minimum dominating sets:

Computing dominating sets:
Dominating sets:
[set([2L]), set([1L, 2L]), set([1L, 3L]),
set([2L, 3L]), set([0L, 3L]), set([0L, 2L]),
 set([0L, 1L, 2L]), set([0L, 2L, 3L]),
 set([1L, 2L, 3L]), set([0L, 1L, 3L])]
Minimum dominating sets:
[set([2L])]
Minimum dominating matrix:
[[0 1 1 0]
 [1 0 1 0]
 [1 1 1 1]
 [0 0 1 0]]
Minimum dominating eigenvalues:
[-1.0, -1.0, 0.38196601125010493,
2.6180339887498945]
Minimum dominating energy of a graph:
[4.999999999999999]

Computing minimum dominating Laplacian energy:

Computing minimum dominating sets:

Computing dominating sets:
Dominating sets:
[set([2L]), set([1L, 2L]), set([1L, 3L]),
set([2L, 3L]), set([0L, 3L]), set([0L, 2L]),
set([0L, 1L, 2L]), set([0L, 2L, 3L]),
set([1L, 2L, 3L]), set([0L, 1L, 3L])]
Minimum dominating sets:
[set([2L])]
Degree matrix:
[[2. 0. 0. 0.]
 [0. 2. 0. 0.]
 [0. 0. 3. 0.]
 [0. 0. 0. 1.]]
Minimum dominating  matrix:
[[0 1 1 0]
 [1 0 1 0]
 [1 1 1 1]
 [0 0 1 0]]
Minimum dominating Laplacian matrix:
[[ 2. -1. -1.  0.]
 [-1.  2. -1.  0.]
 [-1. -1.  2. -1.]
 [ 0.  0. -1.  1.]]
Minimum dominating Laplacian eigenvalues:
[-0.30277563773199445, 0.9999999999999999,
3.0, 3.3027756377319957]
Minimum dominating Laplacian energy of a graph:
[7.60555127546399]

Computing Atom-Bond Connectivity Index:
Atom-Bond Connectivity Index:
2.9378169244873687527

Computing Atom-Bond Connectivity Index2:
Atom bond connectivity index2:
0.0

Computing Atom-Bond Connectivity Index4:
Atom bond connectivity index4:
5.634048107060956

Computing Atom-Bond Connectivity Index5:
atom bond connectivity index5:
0.0
```

Brief summary and conclusion

In this paper, we introduced MathGraph, an open-source and cross-platform Python package. As a Python package, MathGraph is easily integrable with graph visualization softwares. This helps researchers in graph theory to either create a graph programmatically using Python program or draw the graph using Graphical User Interface (GUI) tool such as 'Tulip' to compute distinct sets, energies and topological indices of graphs..

Funding : Not applicable

Competing interests :

The authors declare that they have no competing interests.

Authors contributions :

MRR, SV and MRA drafted the manuscript. RPK and DSN revised it. All authors read and approved the final manuscript.

[1] C. Adiga, A. Bayad, I. Gutman, S. A. Srinivas, The minimum covering energy of a graph. *Kragujevac J. Sci.* **34** (2012), 39-56.

[2] C. Adiga and M. Smitha, On maximum degree energy of a graph, *International Journal of Contemporary Mathematical Sciences*, **4** (2009), No. 8, 385-396.

[3] Calimli M.H., The Fifth Version of Atom Bond Connectivity Index (ABC5) of an infinite class of dendrimers, *Optoelectron. Adv. Mater.- Rapid Commun.,* Vol.5, No.10, 2011, pp.1091-1092.

[4] J. Chen, J. Liu, X. Guo, Some upper bounds for the atom-bond connectivity index of graphs, *Appl. Math. Lett.*, **25** (2012), 1077-1081.

[5] J. Chen, X. Guo, The atom-bond connectivity index of chemical bicyclic graphs, *Appl. Math. j. Chinese Univ.*, **27** (2012), 243-252.

[6] Dragan Stevanović and Alexander Vasilyev, MathChem: A Python Package For Calculating Topological Indices, *MATCH Commun. Math. Comput. Chem.*, **71** (2014), 657-680.

[7] E. Estrada, Atom-bond connectivity and the energetic of branched alkanes, *Chem. Phy. Lett.*, **463** (2008), 422-425.

[8] E. Estrada, L. Torres, L. Rodriguez, I. Gutman, An atom-bond connectivity index: Modelling the enthalpy of formation of alkanes, *Indian Journal of Chemistry*, **37A** (1998), 849-855.

[9] M. R. Farahani, Computing fourth atom-bond connectivity index of V-Phenylenic Nanotubes and Nanotori. *Acta Chimica Slovenica.*, **60(2)**, (2013), 429-432.

[10] M. R. Farahani, On the Fourth atom-bond connectivity index of Armchair Polyhex Nanotube, *Proc. Rom. Acad.*, Series B, **15(1)**, (2013), 3-6.

[11] B. Furtula, A. Gravoc, D. Vukicevic, Atom-bond connectivity index of trees, *J. Math. Chem.*, **48** (2010), 370 - 380.

[12] M. Ghorbani, M. A. Hosseinzadeh, Computing ABC_4 index of Nanostar dendrimers. *Optoelectron. Adv. Mater-Rapid commun.*, **4(9)**, (2010), 1419-1422.

[13] A. Graovac, M. Ghorbani, A new version of atom-bond connectivity index, *Acta. Chim. Slov.*, **57**, (2010), 609-612.

[14] I. Gutman, The energy of a graph, *Ber. Math-Statist. Sekt. Forschungsz.Graz*, **103** (1978), 1-22.

[15] I. Gutman and B. Zhou, Laplacian energy of a graph, *Linear Algebra and Its Applications*, **414** (2006), 29- 37.

[16] X. Li, Y.Shi and I. Gutman, Graph Energy, *Springer*, **1** (2010), 266 .

[17] M. R. Rajesh Kanna, B. N. Dharmendra, and G. Sridhara, Minimum dominating energy of a graph, *International Journal of Pure and Applied Mathematics*, **85** (2013, No. 4, 707-718. [http://dx.doi.org/10.12732/ijpam.v85i4.7]

[18] Willem H. Haemers, Seidel switching and graph energy, *MATCH Communications in Mathematical and in Computer Chemistry*, **68** (2012), 653-659.

[19] R. Xing, B. Zhou, Z. Du, Further results on atom-bond connectivity index of trees, *Discr. Appl. Math.*, **158** (2010), no. 14, 1536-1545.

[20] R. Xing, B. Zhou, F. Dong, On atom-bond connectivity index of connected graphs, *Discr. Appl. Math.*, in press.

[21] R. Xing, B. Zhou, F. Dong, Extremal trees with fixed degree sequence for atom-bond connectivity index, *Filomat*, **26** (2012), 683 - 688.