Introduction to Mathchem – Python package for calculating topological indices.*

Alexander Vasilyev [1] Dragan Stevanović [2]

[1]IAM, University of Primorska

[2]FAMNIT, University of Primorska
PMF, University of Niš

Ljubljana, April 7, 2014

*Supported by Slovenian Research Agency, Young researcher's programme

The Mathchem Project

What is Mathchem

Mathchem is a free open-source Python package for calculating topological indices and other invariants of molecular graphs.

Main features:

- Completely free. No fees, no trial periods
- Open-source. All the sources are available on GitHub.
- Written as a Python package. Python is esay-to-learn language which is widely spread among scientists.
- Cross-platform.
- Could be easily integrated into Sage.
- Could be used with Network X.
- Contains most of the recently introduced topological indices.

What Mathchem is able to calculate

- **Common graph properties:**

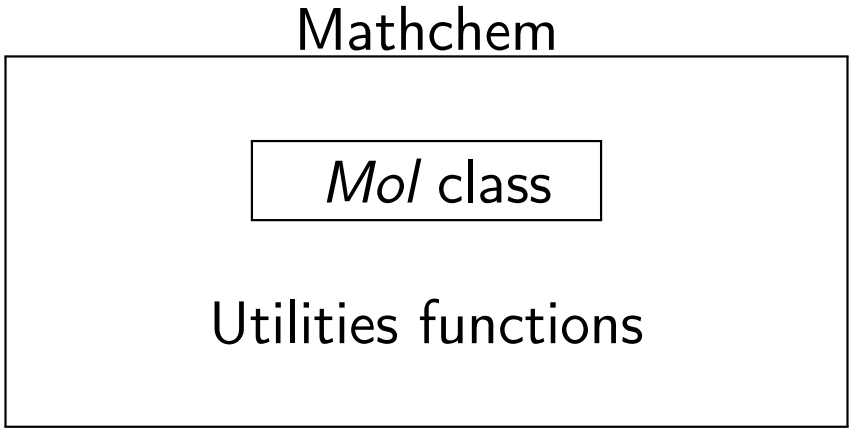
  Order, Size, Diameter, Degrees, Eccentricity, Connectedness.
- **Matrices:**

  Adjacency matrix, Incidence matrix, Distance matrix, Resistance Distance matrix, Reciprocal Distance matrix, Laplacian matrix, Signless Laplacian matrix, Normalized Laplacian matrix, Seidel matrix.
- **Graph spectra:**

  Spectrum, Spectral moments, Spectral radius, Energy, Incidence energy.
- **Topological indices:**

  Zagreb M1 Index, Zagreb M2 Index, Connectivity index, Sum-Connectivity index, Geometric-Arithmetic index, Eccentric Connectivity Index, Randić Index, Atom-Bond Connectivity Index, Estrada Index, Degree Distance, Reverse Degree Distance, Molecular Topological Index, Eccentric Distance Sum, Balaban J index, Sum-Balaban index, Kirchhoff Index, Wiener Index, Terminal Wiener Index, Reverse Wiener Index, Hyper-Wiener Index, Harary Index, Laplacian-like energy, Multiplicative Sum Zagreb index, Multiplicative P1 Zagreb index, Multiplicative P2 Zagreb index, 148 Discrete Adriatic Indices.

Structure



Installation as a Python module

There are two ways to install Mathchem as a Python module.

1. From **Python Package Index** using *pip* tool:

```
1 pip install mathchem
```

2. Using the module built-in *setup.py* installer.

   Download python module archive file from the homepage
   Unpack it and run:

```
1 python setup.py install
```

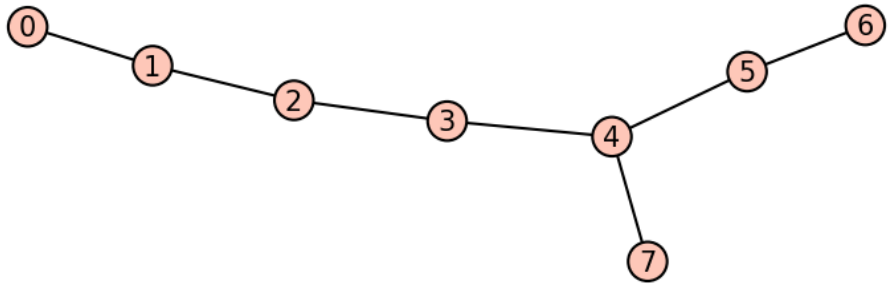
Useful links

Mathchem homepage	mathchem.iam.upr.si
Python Package Index	pypi.python.org
pip installer	pypi.python.org/pypi/pip

Mol class. Initialization.

An instance of the *Mol* class can be initialized with *Graph6* or *Sparse6* string.

Example:



3-methyl-heptane structure

```
1 >>> import mathchem
2 >>> m = mathchem.Mol('GhCGGO')
```

Mol class. Initialization.

A graph can be constructed from the list of edges or from the adjacency matrix.

Edge list example:

```
>>> import mathchem
>>> e = [(3,2),(2,5),(5,3)]
>>> m = mathchem.Mol()
>>> m.read_edgelist(e)
```

Adjacency matrix example:

```
>>> import mathchem
>>> a = [[0,1,1],[1,0,1],[1,1,0]]
>>> m = mathchem.Mol()
>>> m.read_matrix(a)
```

Import data via files

- **SD file (.sdf)**
  `read_from_sdf(fname [, hydrogens = False])`
- **Sybyl 2 Molfile (.mol, .ml2, .mol2)**
  `read_from_mol2(fname [, hydrogens = False])`
- **MDL Molfile (.mol)**
  `read_from_mol(fname [, hydrogens = False])`
- **Graph6 (.g6)**
  `read_from_g6(fname)`
- **Sparse6 (.s6)**
  `read_from_s6(fname)`
- **Planar code (.plc)**
  `read_from_planar_code(fname)`

```
1  >>> import mathchem
2  >>> mols = mathchem.read_from_sdf('octanes.sdf')
3  >>> print len(mols)
4      18
```

Import data from NCI on-line database

The Internet connection is required for the following functions:

- **Search by name**
  `read_from_NCI_by_name(name [, hydrogens = False])`
- **Search by NSC number**
  `read_from_NCI_by_NSC(num [, hydrogens = False])`
- **Search by CAS number**
  `read_from_NCI_by_CAS(num [, hydrogens = False])`

```
1  >>> import mathchem
2  >>> mols = mathchem.read_from_NCI_by_CAS('589-81-1')
3  >>> print mols[0]
4      Molecular graph on 8 vertices
```

Useful links

Enhanced NCI Database Browser cactus.nci.nih.gov/ncidb2.2/

Mol methods. Common graph properties.

- **Order** – `Mol.order()` or `Mol.n()`
- **Size** – `Mol.size()` or `Mol.m()`
- **Diameter** – `Mol.diameter()`
- **Degree of all vertices** – `Mol.degrees()` or `Mol.deg()`
- **Vertices** – `Mol.vertices()`
- **Edges** – `Mol.edges()`
  List of all edges represented by couples of vertices
- **Eccentricity** – `Mol.eccentricity()`
  List of eccentricity value for all vertices
- **Connectedness** – `Mol.is_connected()`
  True if graph is connected or False otherwise

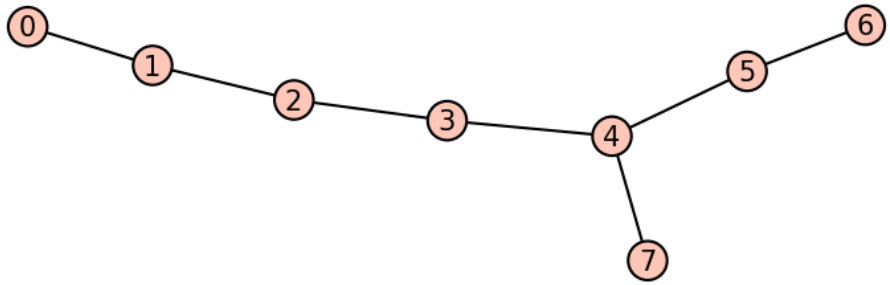
```
1  >>> import mathchem
2  >>> m = mathchem.Mol('GhCGGO')
3  >>> print m.degrees()
4      [1, 2, 2, 2, 3, 2, 1, 1]
```

Mol methods. Matrices.

Matrix	Class method
Adjacency matrix A	`Mol.adjacency_matrix()` or `Mol.A()`
Incidence matrix B	`Mol.incidence_matrix()` or `Mol.B()`
Laplacian matrix $L = Deg - A$	`Mol.Mol.laplacian_matrix()` or `Mol.L()`
Signless Laplacian matrix $Q = Deg + A$	`Mol.signless_laplacian_matrix()` or `Mol.Q()`
Normalized Laplacian matrix N $[N]_{i,j} = \frac{1}{\sqrt{deg(v_i)deg(v_j))}} l_{i,j}$	`Mol.normalized_laplacian_matrix()` or `Mol.NL()`
Distance matrix D	`Mol.distance_matrix()` or `Mol.D()`
Resistance Distance matrix Ω	`Mol.resistance_distance_matrix()`
Reciprocal Distance matrix rD	`Mol.reciprocal_distance_matrix()`

Mol methods. Matrices.

Matrix example:



3-methyl-heptane structure

```
>>> import mathchem
>>> m = mathchem.Mol('GhCGGO')
>>> print m.distance_matrix()
    [[0 1 2 3 4 5 6 5]
     [1 0 1 2 3 4 5 4]
     [2 1 0 1 2 3 4 3]
     [3 2 1 0 1 2 3 2]
     [4 3 2 1 0 1 2 1]
     [5 4 3 2 1 0 1 2]
     [6 5 4 3 2 1 0 3]
     [5 4 3 2 1 2 3 0]]
```

Mol methods. Graph spectra.

- **Spectrum** – `spectrum(matrixname)`,
  where the argument `matrixname` is:
  - `'adjacency'` or `'A'` (by default) Adjacency matrix
  - `'laplacian'` or `'L'` Laplacian matrix
  - `'distance'` or `'D'` Distance matrix
  - `'signless_laplacian'` or `'Q'` Signless Laplacian matrix
  - `'normalized_laplacian'` or `'NL'` Normalized Laplacian matrix
  - `'resistance_distance'` or `'RD'` Resistance Distance matrix
  - `'reciprocal_distance'` Reciprocal Distance matrix
- **Spectral moments** – `spectral_moment(k, matrixname)`
- **Graph energy** – `energy(matrixname)`
- **Incidence energy** – `incidence_energy()`
- **Spectral radius** – `spectral_radius(matrixname)`

Mol methods. Graph spectra.

Example:

```
>>> import mathchem
>>> m = mathchem.Mol('GhCH?_')
>>> m.spectrum()
  [2.095293985223914, 1.355674293978083, 0.737640305228187 2,
   0.4772599964740198, -0.4772599964740197, -0.737640305228187 4,
  -1.355674293978082 4, -2.095293985223914]

>>> m.spectrum('distance')
  [17.67586981788181 8, -0.4268447865902264, -0.5999662634461097,
  -0.8565662710452482, -1.460624478516444 8, -2.744728088663583,
  -3.615279075919263, -7.971860853700944]
```

Topological indices

Zagreb M1 Index	`Mol.zagreb_m1_index()`
Zagreb M2 Index	`Mol.zagreb_m2_index()`
Connectivity index (R)	`Mol.connectivity_index(power)`
Sum-Connectivity index	`Mol.sum_connectivity_index()`
Geometric-Arithmetic index	`Mol.geometric_arithmetic_index()`
Eccentric Connectivity Index	`Mol.eccentric_connectivity_index()`
Randic Index	`Mol.randic_index()`
Atom-Bond Connectivity Index (ABC)	`Mol.atom_bond_connectivity_index()`
Estrada Index (EE) for all matrices	`Mol.estrada_index(matrix)`
Degree Distance (DD)	`Mol.degree_distance()`
Reverse Degree Distance (rDD)	`Mol.reverse_degree_distance()`
Molecular Topological Index (MTI)	`Mol.molecular_topological_index()`
Eccentric Distance Sum	`Mol.eccentric_distance_sum()`
Balaban J index	`Mol.balaban_j_index()`
Sum-Balaban index	`Mol.sum_balaban_index()`
Kirchhoff Index (Kf) or Resistance	`Mol.kirchhoff_index()` = `Mol.resistance()`
Wiener Index (W)	`Mol.wiener_index()`
Terminal Wiener Index (TW)	`Mol.terminal_wiener_index()`
Reverse Wiener Index (RW)	`Mol.reverse_wiener_index()`
Hyper-Wiener Index (WW)	`Mol.hyper_wiener_index()`
Harary Index (H)	`Mol.harary_index()`
Laplacian-like energy (LEL)	`Mol.LEL()`
log(Multiplicative Sum Zagreb index)	`Mol.multiplicative_sum_zagreb_index()`
log(Multiplicative P1 Zagreb index)	`Mol.multiplicative_p1_zagreb_index()`
log(Multiplicative P2 Zagreb index)	`Mol.multiplicative_p2_zagreb_index()`
148 Discrete Adriatic Indices	`Mol.adriatic_index(p,i,j,a)` or by name:
	`Mol.max_min_rodeg_index()`
	`Mol.randic_type_sdi_index()`
	...

lazy calculations

For performance reasons Mathchem calculates resource consuming data, such as *eigenvalues*, *energy*, *distance matrices* etc. on demand and then saves the results.
Example:

```
>>> import mathchem
>>> m = mathchem.Mol('GhCH?_')
>>> print m.degree_distance(), m.balaban_j_index()
    216  3.29247815608
>>> m.wiener_index()
    68
```

In the example above the *distance matrix* will be calculated only once – before calculating the first index.

The Sage Project

Mission Statement

Create a viable free open source alternative to Magma, Maple, Mathematica, and Matlab

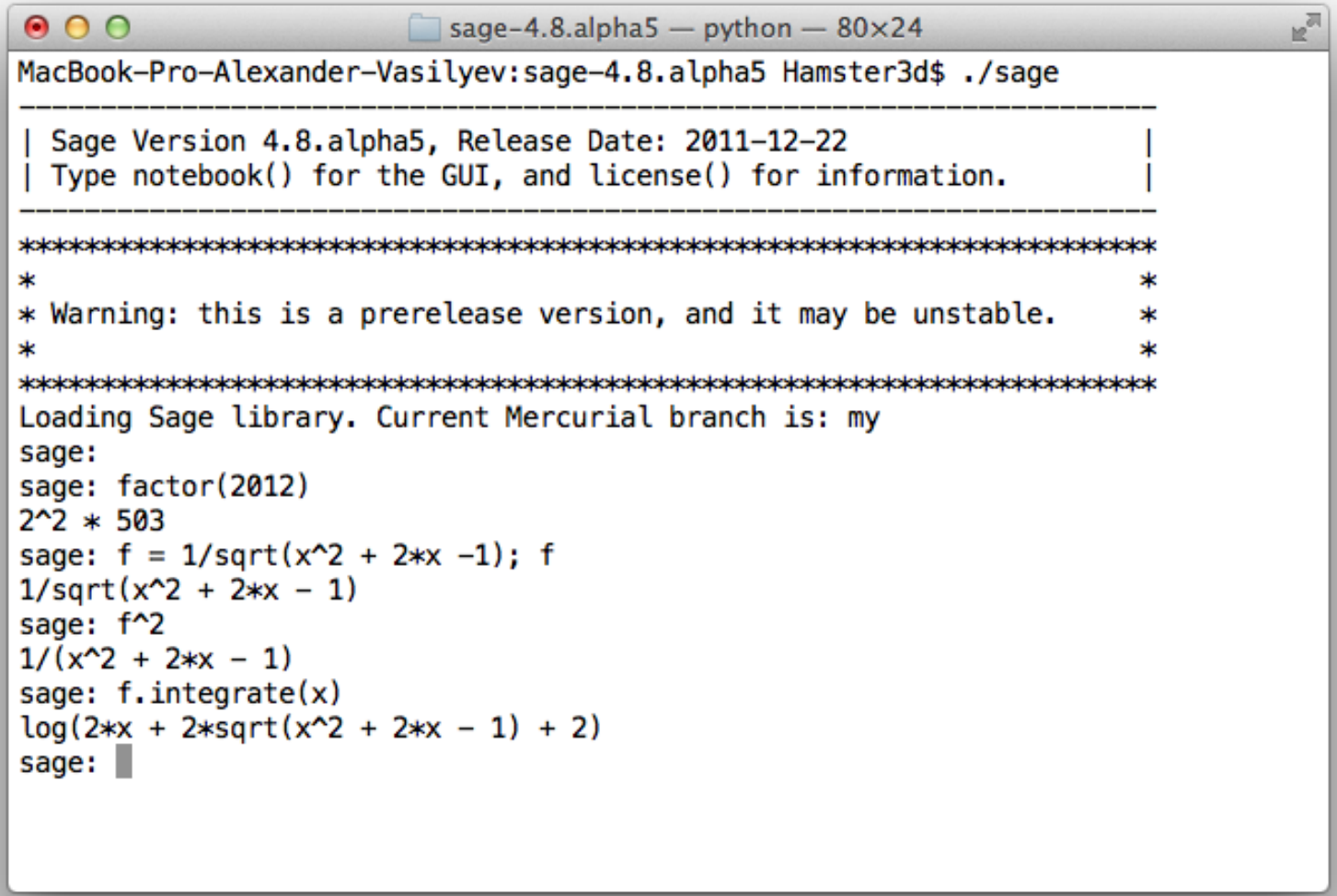
- The mathematical features of Magma, Maple, Mathematica, and Matlab with comparable speed.
- Beautiful interactive 2d and 3d graphics.
- A notebook interface and an IDE.

What is Sage

1. **A self-contained distribution** of over 90 open source packages that is easy to build from source.
2. **Interfaces** that smoothly tie together all these libraries and packages.
3. **A new library** that implements novel algorithms. About a half million lines of code written by a worldwide community of about 200 people over the last 6 years.

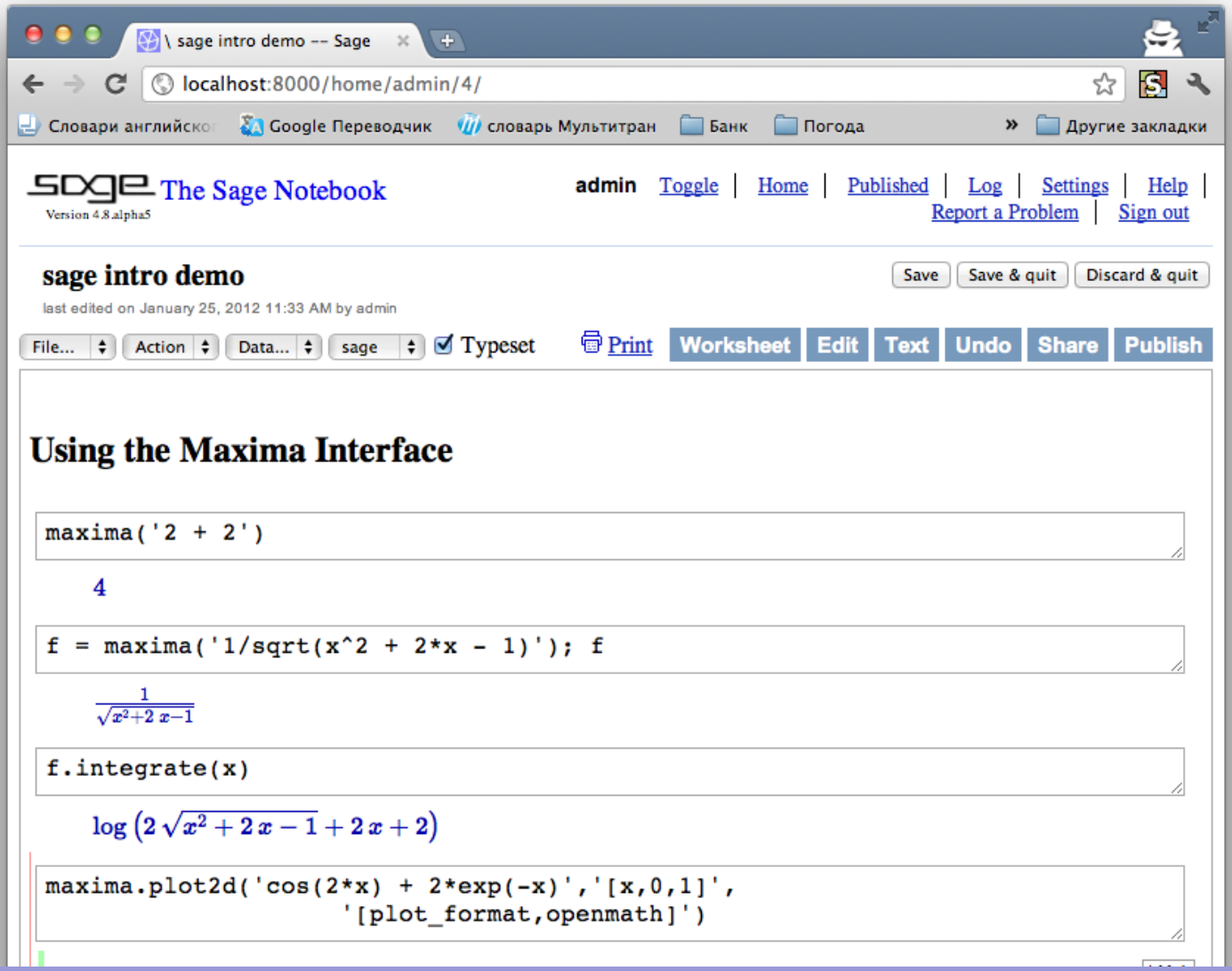
Sage in use

Command Line Sage



Sage in use

Sage notebook



Try Sage

Sage is crossplatform. Supported platforms:

- Linux
- Apple Mac OS X
- Solaris
- Microsoft Windows (run on virtual machine with Linux)
- Live CD

The best way to try Sage is using on-line Sage notebook.

Useful links

Sage homepage	www.sagemath.org
Sage Notebook	www.sagenb.org
Sage documentation	www.sagemath.org/help.html
Bug report and contribution	trac.sagemath.org/sage_trac/report

Mathchem installation as a Sage module

1. Download **spkg** file from the Mathchem homepage.
2. Save it into your **Sage** directory
3. Run Sage with the command to install a new package:

```
1  sage -f mathchem-1.0.5.spkg
```

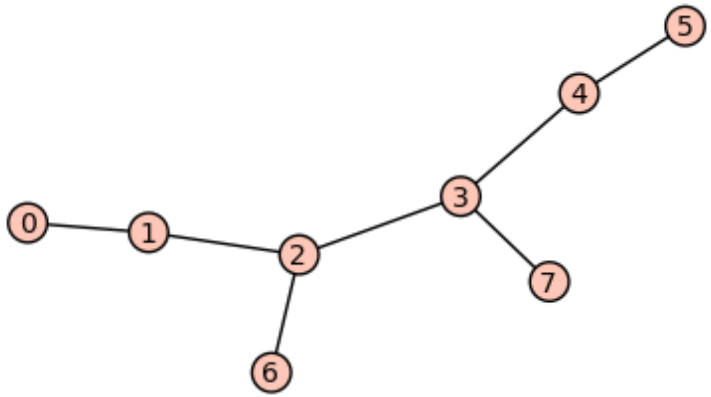
The official instructions on installing additional packages for Sage can be found on the project homepage http://www.sagemath.org

Integration with Sage. Mathchem $\longrightarrow$ Sage.

Sage has its own **Graph** class. Mathchem **Mol** class has a built-in method `Mol.sage_graph()` which returns Sage Graph class instance .

In the following example we convert Mathchem molecular graph to Sage Graph, visualize it and check for hamiltonicity.

```
1  sage: import mathchem
2  sage: m = mathchem.Mol('GhCH?_')
3  sage: g = m.Sage_graph()
4  sage: g.is_hamiltonian()
5      False
6  sage: g.show()
```



Integration with Sage. Sage $\longrightarrow$ Mathchem.

On the other hand, there is an easy way to use a Sage graph as a source for *Mol* instance using a method of Sage Graph class `Graph.graph6_string()`. In the following example we create a random graph with 10 vertices and 20 edges and then convert it to *Mol* class instance.

```
1  sage: import mathchem

2  sage: g = graphs.RandomGNM(10,20)
3  sage: m = mathchem.Mol(g.graph6_string())
4  sage: m.randic_index()
5      4.836902752369998
```

Integration with Network X. Mathchem $\longrightarrow$ Network X

NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

For further calculations outside of Mathchem package the *Mol* class instance can be easily converted to Network X graph.

```
>>> import mathchem
>>> import networkx

>>> m = mathchem.Mol('GhCH?_')
>>> g = m.NX_graph()
>>> networkx.is_bipartite(g)
    True
```

Useful links

Network X homepage networkx.github.io

Integration with Network X. Network X $\longrightarrow$ Mathchem

Any Network X graph can be converted to Mathchem for further calculations using `read_edgelist` or `read_edgelist` methods of the Mol class.

```
>>> import mathchem
>>> import networkx

>>> b = networkx.complete_bipartite_graph(3,5)
>>> m = mathchem.Mol()
>>> m.read_edgelist(b.edges())
```

Example 1. Loading data.

Let us start Sage in terminal mode and import Mathchem.

```
sage: import mathchem
```

Here we import some data from NCI database. We would like to retrieve first 1000 of molecules.

```
sage: mols = mathchem.read_from_NCI_by_NSC('1-1000')
sage: len(mols)
    993
```

The actual number of retrieved records is 993 because the database has gaps.

Now we want to have only connected graphs. Here we utilize standard Python `filter(function, array)` function.

```
sage: mols_c = filter(lambda m: m.is_connected(), mols)
sage: len(mols_c)
    980
```

In the code above we used a lambda-function which allows to create functions on-the-fly.

Example 1. Loading data.

Let us start Sage in terminal mode and import Mathchem.

```
sage: import mathchem
```

Here we import some data from NCI database. We would like to retrieve first 1000 of molecules.

```
sage: mols = mathchem.read_from_NCI_by_NSC('1-1000')
sage: len(mols)
    993
```

The actual number of retrieved records is 993 because the database has gaps. Now we want to have only connected graphs. Here we utilize standard Python `filter(function, array)` function.

```
sage: mols_c = filter(lambda m: m.is_connected(), mols)
sage: len(mols_c)
    980
```

In the code above we used a lambda-function which allows to create functions on-the-fly.

Example 1. Calculations.

Now we want to explore correlations between two indices. First we create two lists:

```
sage: m1 = [m.zagreb_m1_index() for m in mols_c]
sage: m2 = [m.zagreb_m2_index() for m in mols_c]
```

Now we use standard Python function `zip` to join two lists. For two lists $A = [a_1, a_2, ...a_n]$ and $B = [b_1, b_2, ...b_n]$ the resulting list will be $zip(A, B) = [(a_1, b_1), (a_2, b_2), ..., (a_n, b_n)]$. Finally we draw a scatter plot using Sage functionality.

```
sage: scatter_plot( zip(m1,m2) )
```

The resulting output is shown on the next slide.

Example 1. Calculations.

Now we want to explore correlations between two indices. First we create two lists:

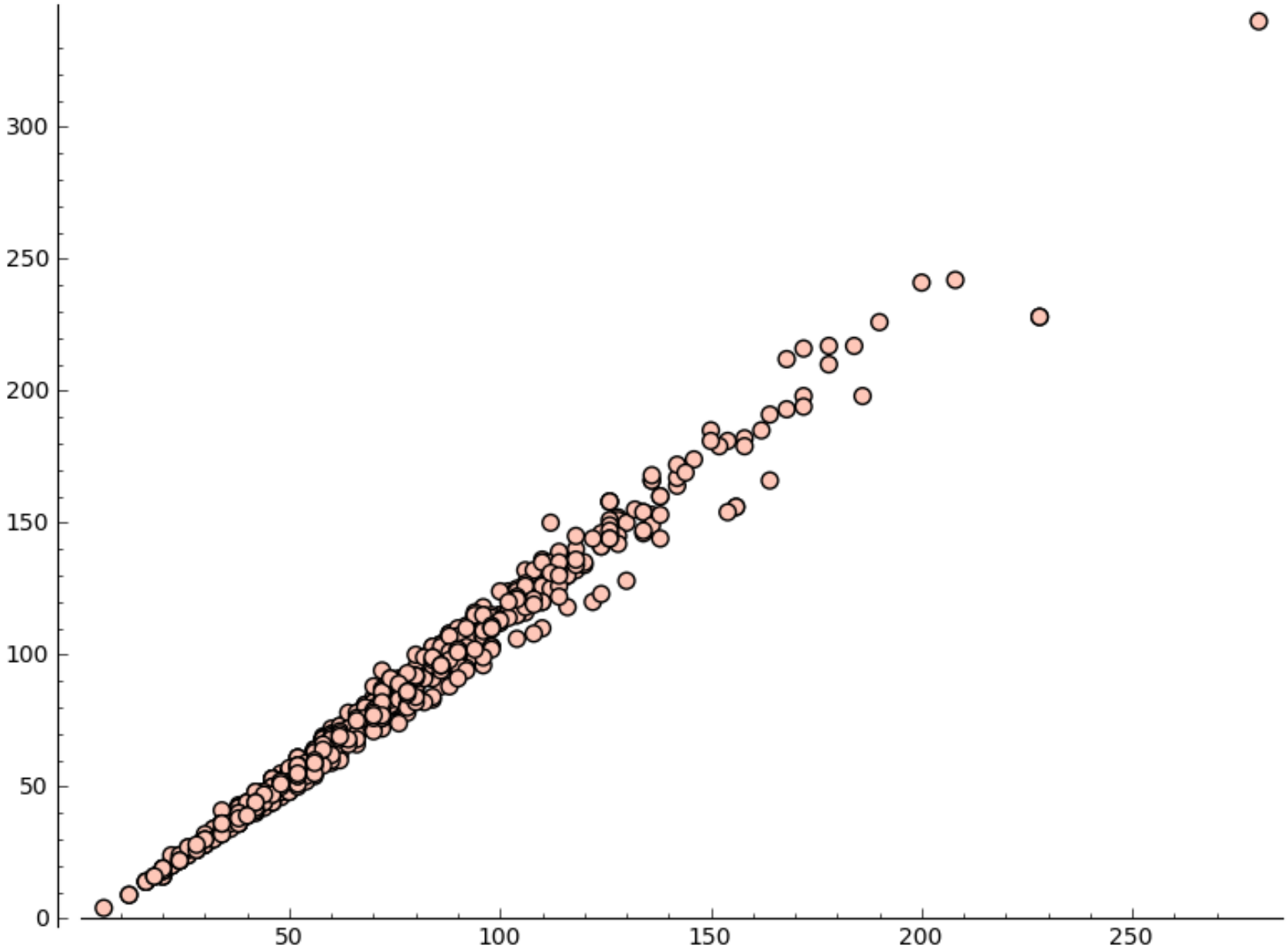
```
sage: m1 = [m.zagreb_m1_index() for m in mols_c]
sage: m2 = [m.zagreb_m2_index() for m in mols_c]
```

Now we use standard Python function `zip` to join two lists.
For two lists $A = [a_1, a_2, ...a_n]$ and $B = [b_1, b_2, ...b_n]$ the resulting list will be $zip(A, B) = [(a_1, b_1), (a_2, b_2), ..., (a_n, b_n)]$. Finally we draw a scatter plot using Sage functionality.

```
sage: scatter_plot( zip(m1,m2) )
```

The resulting output is shown on the next slide.

Example 1. Scatter plot.



Example 2. Interactive widget.

Now we will use Sage in notebook mode to show an example of how to create an interactive tool to explore correlations between two indices.

First run Sage in notebook mode:

```
1 sage --notebook
```

Sage starts a local web server and automatically opens a new browser's window with admin page.

Let us create a new worksheet and start writing a program.

Example 2. Loading data.

We will again use NCI database as a data source.

Here we load data fom the database and apply filter to obtain only connected graphs.

```
1  import mathchem
2  mols = mathchem.read_from_NCI_by_NSC('1-5000')
3  mols_c = filter(lambda m: m.is_connected(), mols)
4  print 'Test set of ', len(mols_c), 'compounds'
5      Test set of  4800 compounds
```

Example 2.

Now we want to create a visual widget where we can select two indices and automatically build a scatter plot and calculate correlation between them.

Let us first define a list of the indices we are interested in:

```
methods = ['order', 'diameter', 'balaban_j_index', 'incidence_energy',
'zagreb_m1_index', 'zagreb_m2_index', 'eccentric_connectivity_index',
'randic_index', 'atom_bond_connectivity_index', 'estrada_index',
'degree_distance', 'reverse_degree_distance', 'kirchhoff_index',
'eccentric_distance_sum', 'energy', 'molecular_topological_index',
'wiener_index', 'terminal_wiener_index', 'reverse_wiener_index',
'hyper_wiener_index', 'harary_index', 'LEL', 'randic_type_lodeg_index',
'randic_type_sdi_index', 'randic_type_hadi_index', 'sum_lordeg_index']
```

This is the list of some methods of *Mol* class which return the value of corresponding index or property.

Now we need to include a **ScyPy** statistics library for using linear regress.

```
1  import scipy.stats as stats
```

Now we are ready to write an interactive widget:

```
1  @interact
2  def index_correlations(index_A = selector(methods,label="Index A"), \
3      index_B = selector(methods,label="Index B")):
4      data_A = [getattr(m, index_A)() for m in mols_c]
5      data_B = [getattr(m, index_B)() for m in mols_c]
6      data = zip(data_A, data_B)
7      slope, intercept, r, ttprob, stderr = stats.linregress(data)
8      print 'Correlation coefficient: ', r**2
9      canvas = scatter_plot(data)
10     canvas += plot(slope*x+intercept,min(data_A),max(data_A))
11     canvas.show(figsize=[10,4], axes_labels=[index_A, index_B])
```

Now we need to include a **ScyPy** statistics library for using linear regress.

```
1  import scipy.stats as stats
```

Now we are ready to write an interactive widget:

```
1  @interact
2  def index_correlations(index_A = selector(methods,label="Index A"), \
3      index_B = selector(methods,label="Index B")):
4      data_A = [getattr(m, index_A)() for m in mols_c]
5      data_B = [getattr(m, index_B)() for m in mols_c]
6      data = zip(data_A, data_B)
7      slope, intercept, r, ttprob, stderr = stats.linregress(data)
8      print 'Correlation coefficient: ', r**2
9      canvas = scatter_plot(data)
10     canvas += plot(slope*x+intercept,min(data_A),max(data_A))
11     canvas.show(figsize=[10,4], axes_labels=[index_A, index_B])
```

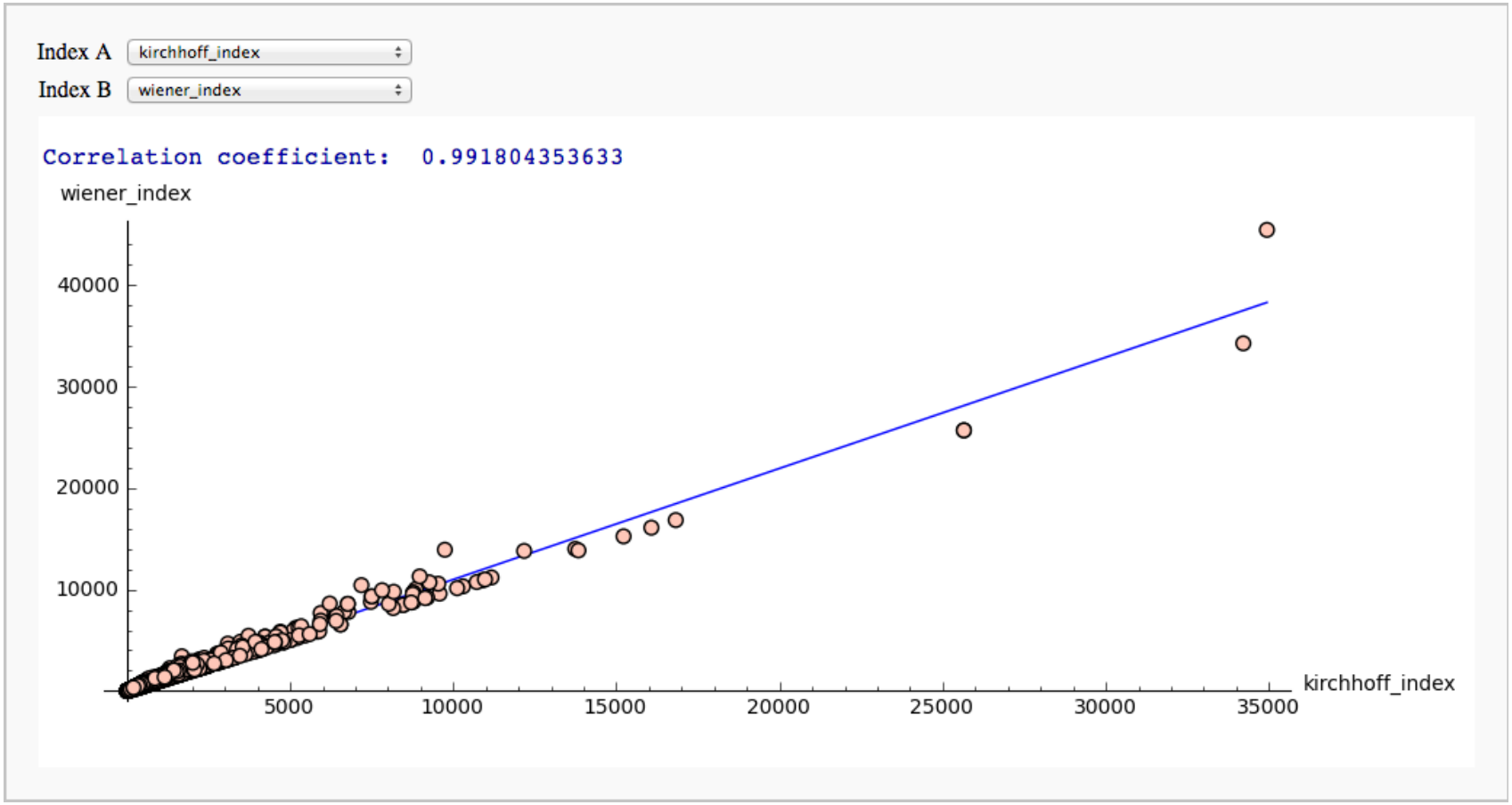


Figure: Interactive widget.

Thank you!