

Graph Theory with Lab4IQM

Chapter 1: fundamental concepts

Lab4IQM Graph Theory module 0.2 · lab4iqm.com

This companion follows D. B. West, *Introduction to Graph Theory* (2nd edition, Prentice Hall, 2001), chapter by chapter. For each section it gives the Lab4IQM commands that draw the graphs and run the algorithms of the section, with the book's section and theorem numbers. Read the book for the proofs; use Lab4IQM to see the graphs and to test conjectures on examples. Exercises are numbered as in the book (and its solution manual); the table at the end of each chapter says which commands help with which exercises. Type the commands in the Lab4IQM console after

```
import graphs
```

`help(graph)` lists the ways to name a graph and the drawing options; the Examples panel has one example per section. In a Python script, `import graphs` gives the same commands as functions, e.g. `graphs.graph("petersen", layout="circle")`.

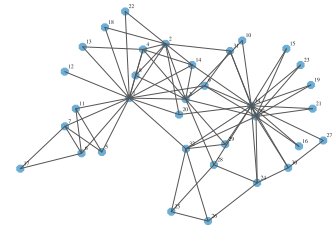
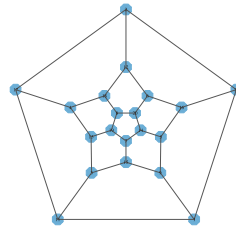
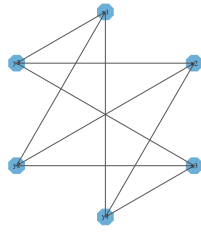
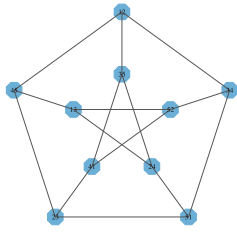
Naming and drawing a graph

Every command takes a graph as its first argument. Put it in quotes unless it is a single word such as `petersen`, `K5`, `C7` or `Q3`.

families	K5, "K(3,3)", "K(2,3,4)", C7, P5 (n vertices), W6, Q4, "star(5)", "empty(4)", "grid(3,4)", "kneser(5,2)", "circulant(8,1,3)", "turán(8,3)"
named graphs	petersen, heawood, grotzsch, konigsberg, the five Platonic solids (cube, dodecahedron, ...), bull, claw, paw, house, diamond
edges	"a-b, b-c, c-a"; a path or cycle "1-2-3-4-1"; loops "a-a"; repeated edges are kept (multigraphs); arcs "u->v, v<-w, a<->b"; weights "a-b:2.5"; an isolated vertex is a name alone
operations	complement(G), line(G), union(G,H) (disjoint), join(G,H), product(G,H) (Cartesian), mycielski(G), delete(G, v, ...), deledge(G, a-b), subgraph(G, v, ...) (induced), underlying(G), simple(G), reverse(G), directed(G)
random	"gnp(n,p,seed)", "gnm(n,m,seed)", "tree(n,seed)" (uniform, from a Prüfer code), "regular(n,d,seed)", "tournament(n,seed)"
digraphs	"debruijn(k,n)", "transitive(n)", "dcycle(n)"
files	"edges.csv" (an edge list with or without a header source,target,weight, or an adjacency matrix), .edgelist, .adjlist, .graphml, .gml; "graphs:karate.csv" and "graphs:florentine.csv" come with the module

The drawing options are those of NetworkX: `layout=circle`, `shell`, `spring` (Fruchterman-Reingold), `stress` (Kamada-Kawai style: graph distances kept as well as possible), `spectral`, `bipartite`, `tree`, `random`; the default uses the graph's own drawing when it has one (the Petersen graph as a pentagon and a pentagram, the Platonic solids by Tutte's barycentric method, which gives a plane drawing) and `stress` otherwise. `d3=true` draws in 3D, with the Platonic solids and Q_3 as polyhedra. `labels=`, `weights=`, `color=` and `seed=` do what they say. `graphs.save(G,`

"file.graphml") writes a graph that NetworkX reads with `nx.read_graphml`; `.csv`, `.edgelist`, `.adjlist` and `.gml` work too. In an adjacency matrix file, whole numbers are numbers of edges; other numbers are weights.

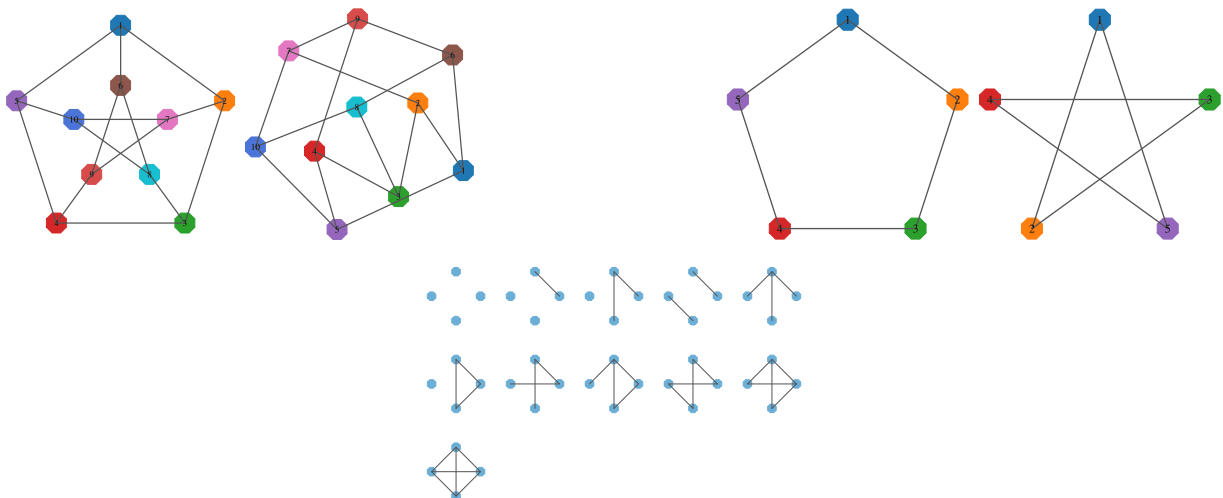


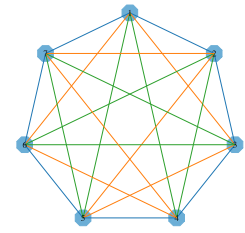
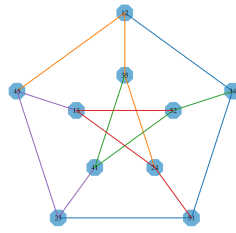
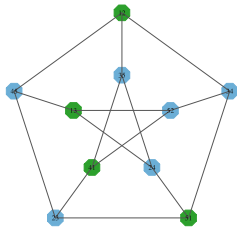
Chapter 1: Fundamental Concepts

1.1 What is a graph?

<code>graph(petersen)</code>	order, size, degrees, girth...
<code>graph("a-b, b-c, c-a, c-d, d-e, e-c")</code>	from its edges
<code>graph(konigsberg)</code>	a multigraph: the seven bridges
<code>matrices("a-b, b-c, c-a, c-d", show=incidence)</code>	adjacency and incidence matrices
<code>isomorphic(petersen, "kneser(5,2)")</code>	the bijection, drawn in colours
<code>isomorphic(C5, "complement(C5)")</code>	self-complementary
<code>isomorphic(C6, "union(C3, C3)")</code>	not isomorphic: why
<code>automorphisms(petersen)</code>	$ \text{Aut} = 120$, one orbit
<code>allgraphs(n=4)</code>	the 11 graphs on four vertices
<code>cliques(petersen, show=independent)</code>	$\omega = 2$, $\alpha = 4$
<code>decompose(petersen, P4)</code>	five copies of P_4 (Ex. 1.1.34)
<code>decompose(K7, C7)</code>	three Hamiltonian cycles (Ex. 1.1.33)

`graph` prints the facts of the section: whether the graph is simple, its order n and size m , the degrees, the components, whether it is bipartite and its girth. The Petersen graph is built as in the book, on the 2-subsets of $\{1, \dots, 5\}$ with disjoint pairs adjacent, so `isomorphic(petersen, "kneser(5,2)")` returns the identity up to the order of the names. `isomorphic` looks for an isomorphism by refining the vertex classes (by degree, then by the degrees of the neighbours, and so on) and backtracking; when there is none it names the first invariant that differs. The two graphs are drawn side by side, and each vertex and its image have the same colour and number. `automorphisms` counts the automorphisms by the orbit–stabiliser theorem and colours the vertices by orbit: vertex-transitive graphs (Petersen, K_n , C_n , Q_k) come out in one colour. `allgraphs(n=...)` draws one graph from each isomorphism class (1, 2, 4, 11, 34, 156 for $n = 1, \dots, 6$; `connected=true`, `edges=m`): the pictures for the book’s question of how many graphs have four vertices. `automorphisms` also gives the edge orbits, so it tells whether a graph is edge-transitive (the claw is, without being vertex-transitive: Exercises 1.1.46–1.1.47). `cliques` finds a largest clique and a largest independent set. `decompose(G, H)` searches for a decomposition of G into copies of H and colours the copies; when there is none it gives the counting reason (the size of H must divide that of G ; copies of a graph with even degrees need even degrees) or reports that the exhaustive search found none, as for the Petersen graph into claws (Exercise 1.1.38).

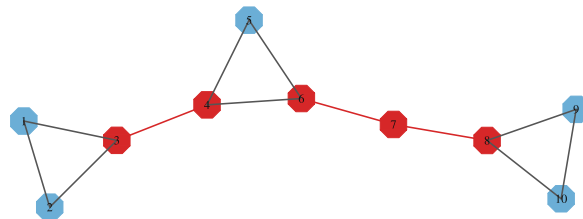
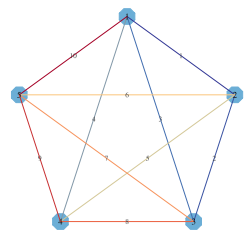
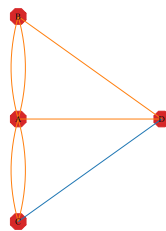
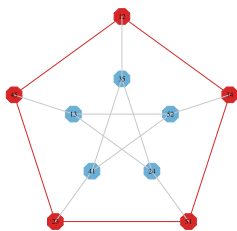


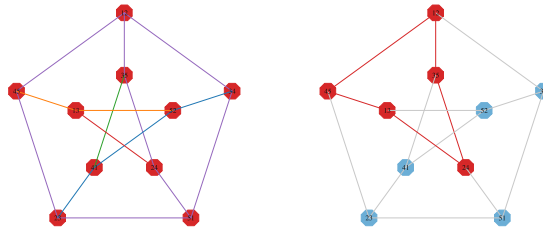


1.2 Paths, cycles and trails

<code>components("union(C3, P4, K(1,3))")</code>	components in colours
<code>bipartite(Q3)</code>	a bipartition
<code>bipartite(petersen)</code>	an odd cycle (Thm. 1.2.18)
<code>euler(konigsberg)</code>	no Eulerian circuit: why
<code>euler(K5)</code>	Hierholzer's algorithm (Thm. 1.2.26)
<code>cuts("1-2-3-1, 3-4, 4-5-6-4, 6-7, 7-8, 8-9-10-8")</code>	cut-vertices and cut-edges
<code>graph(petersen, highlight="12-34-51-23-45-12")</code>	a cycle in red
<code>euler(petersen)</code>	five trails (Thm. 1.2.33)
<code>cycles(petersen)</code>	cycles of each length
<code>bipartite("transpositions(4)")</code>	Exercise 1.2.27

A graph is bipartite if and only if it has no odd cycle (König, Thm. 1.2.18): `bipartite` either draws the two parts or draws an odd cycle in red, found by breadth-first search (an edge between two vertices at the same distance from the root closes an odd cycle). A connected graph has an Eulerian circuit if and only if every vertex has even degree (Thm. 1.2.26); with exactly two odd vertices there is an Eulerian trail between them. `euler` builds the circuit as in the proof (Hierholzer: follow unused edges until stuck, then splice in the closed walks left over), colours the edges from blue (first) to red (last) and numbers them. For Königsberg it marks the four odd vertices. `cuts` finds cut-vertices and cut-edges in one depth-first search; an edge is a cut-edge if and only if it lies on no cycle (Thm. 1.2.14). When a connected graph has $2k > 2$ odd vertices, `euler` draws it decomposed into k trails (Thm. 1.2.33: join a new vertex to the odd vertices, follow an Eulerian circuit, cut it at the new vertex); the Petersen graph needs five (Exercise 1.2.9). `cycles(G)` counts the cycles of each length and gives the girth and circumference; `cycles(G, length=k)` draws one: the Petersen graph has twelve 5-cycles and no 7-cycle (Exercises 1.3.30 and 1.1.25).



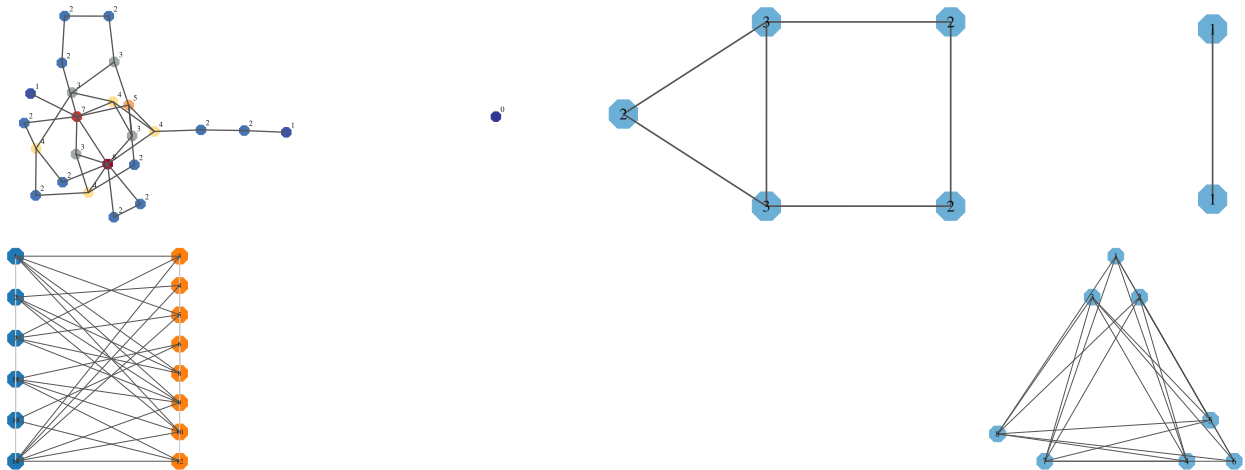


1.3 Vertex degrees and counting

```
degrees("gnp(25, 0.15, 3)")
matrices(K4, power=3)
graphic(d=(3, 3, 2, 2, 2, 1, 1))
graphic(d=(4, 4, 4, 1, 1))
bipartite("gnp(14, 0.45, 2)", sub=true)
graph("turan(8,3)")
deck(kite)
```

```
degrees, degree-sum formula
A^3 counts walks; triangles = trace/6
Havel-Hakimi (Thm. 1.3.31), a realization
not graphic: where it fails
a bipartite subgraph with  $\geq m/2$  edges
the Turan graph
the vertex-deleted subgraphs (Ex. 1.3.36)
```

degrees labels each vertex with its degree and checks $\sum d(v) = 2m$, so the number of odd vertices is even. **graphic** runs the Havel–Hakimi test step by step and draws the simple graph that the steps build when the list is graphic; it also checks the Erdős–Gallai inequalities. **bipartite**(*G*, *sub=true*) follows the proof of the theorem that every loopless graph has a bipartite subgraph with at least half its edges (Thm. 1.3.19): move a vertex with more neighbours on its own side to the other side; each move adds edges between the parts, so the process stops, and then every vertex has at least half its edges to the other side. **graph**("turan(*n*,*r*)") draws the complete *r*-partite graph with parts as equal as possible, the extremal graph of the section's results on graphs with no large cliques (Mantel's theorem for triangles, $r = 2$: Thm. 1.3.23). **deck**(*G*) draws the vertex-deleted subgraphs $G - v$, groups them into isomorphism classes, recovers $m(G)$ as $\sum m(G - v)/(n - 2)$ and, for $n \leq 7$, lists every graph with the same deck (one, as the Reconstruction Conjecture predicts).

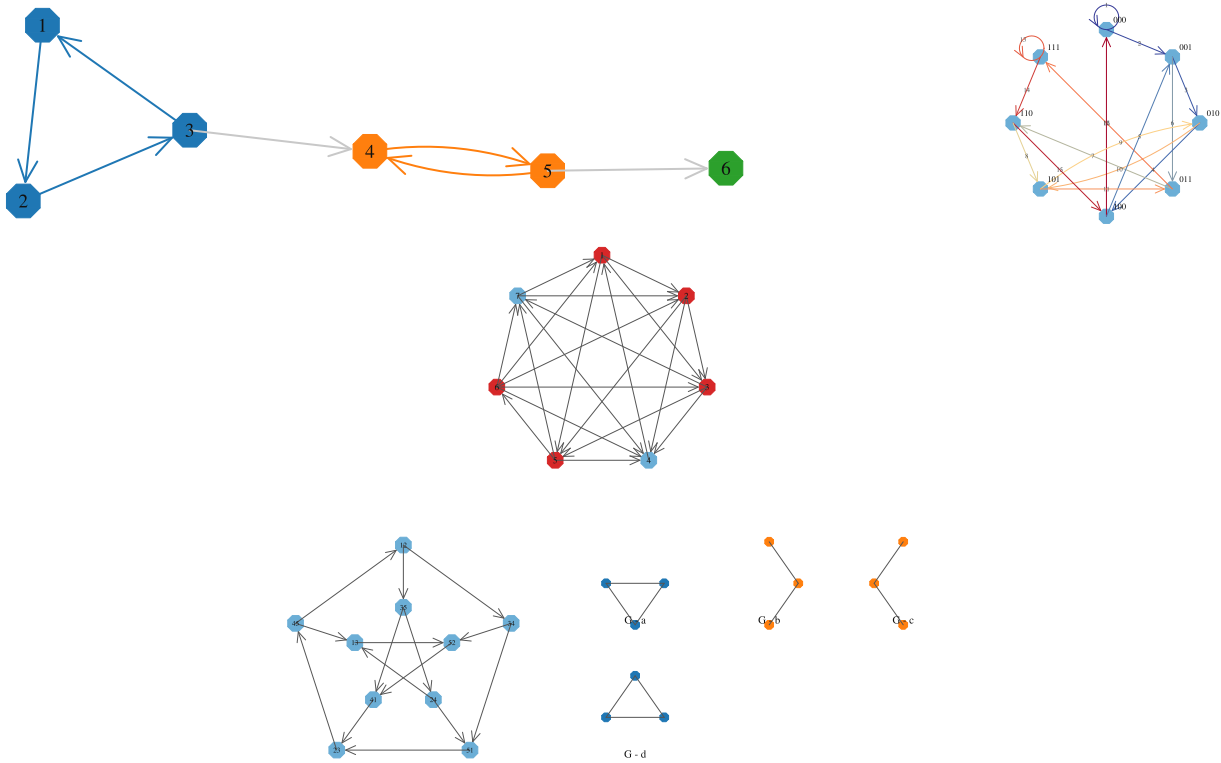


1.4 Directed graphs

```
graph("u->v, v->w, w->u, w->x")
components("1->2, 2->3, 3->1, 3->4, 4->5, 5->4, 5->6")
euler("debruijn(2,4)")
kings("tournament(7, 2)")
orient(petersen)
graphic(p=(3, 2, 2), q=(2, 2, 2, 1))
```

```
a digraph: arrows, in- and out-degrees
strong components
de Bruijn digraph and sequence
kings of a tournament
 $|d_+ - d_-| \leq 1$  (Ex. 1.4.23)
bigraphic lists (Ex. 1.4.32)
```

Arcs are drawn with arrowheads; two opposite arcs bend apart. `components` gives the strong components of a digraph, in the order of the condensation (the first has no entering arc), and greys the arcs between them. The de Bruijn digraph $D(k, n)$ has the words of length $n - 1$ over k letters as vertices, with an arc for each word of length n . Every vertex has in-degree and out-degree k , so it has an Eulerian circuit, and `euler` reads the de Bruijn sequence off it (for $k = 2, n = 4$: a cyclic word of length 16 containing every binary word of length 4 once). A tournament is an orientation of a complete graph. Every tournament has a king, and a vertex of maximum out-degree is one (Prop. 1.4.30). `kings` draws the kings in red, runs the king-finding algorithm of Exercise 1.4.37 step by step, and gives a spanning path of the tournament. For an acyclic digraph `components` gives an order in which every edge goes forward (Exercise 1.4.14). `orient(G)` orients a graph so that out-degree and in-degree differ by at most one everywhere, by the Eulerian-circuit argument of Exercise 1.4.23.



Exercises of Chapter 1

West's exercises ask for proofs; the commands `find` and `test` the examples and counterexamples.

1.1.2–1.1.3	<code>matrices(P3, show=incidence), matrices("K(2,3)").</code>
1.1.10, 1.1.15	<code>graph("complement(G)");</code> components on it.
1.1.11	<code>cliques(G)</code> and <code>cliques(G, show=independent)</code> .
1.1.12–1.1.13, 1.1.24	<code>graph(petersen)</code> (girth, bipartite), <code>bipartite(Q4)</code> , <code>isomorphic(petersen, "kneser(5,2)").</code>
1.1.16–1.1.23	<code>isomorphic(G, H)</code> with your graphs typed as edge lists; <code>allgraphs</code> for the smallest examples.
1.1.25–1.1.28	<code>cycles(petersen); graph("odd(3)")</code> (girth 6).
1.1.31–1.1.39, 1.2.43	<code>decompose(G, H):</code> e.g. <code>decompose(K7, C7)</code> , <code>decompose(petersen, P4)</code> , <code>decompose(K6, "star(3)")</code> , <code>decompose(petersen, "star(3)")</code> , <code>decompose(Q3, P3)</code> ; self-complementary graphs with <code>isomorphic(G, "complement(G)").</code>
1.1.40–1.1.47	<code>automorphisms(G):</code> orders, vertex and edge orbits.
1.2.2, 1.2.30	<code>matrices(K4, power=k)</code> counts walks.
1.2.7, 1.2.25– 1.2.28	<code>bipartite(G)</code> , <code>bipartite("transpositions(4)")</code> , <code>bipartite(G, sub=true)</code> .
1.2.8–1.2.12, 1.2.32–1.2.35	<code>euler(G): euler("K(4,6)")</code> , <code>euler(petersen)</code> (five trails).
1.2.17	<code>graph("adjtranspositions(4)")</code> : connected and 3-regular.
1.2.20–1.2.21	<code>cuts(G)</code> and <code>cuts("complement(G)").</code>
1.3.4–1.3.5, 1.3.23–1.3.29	<code>graph(Q4)</code> , <code>automorphisms(Q3)</code> ($2^k k!$), <code>graph("middle(2)").</code>
1.3.8, 1.3.57– 1.3.63	<code>graphic(d=(...))</code> : <code>graphic(d=(5,5,5,4,2,1,1,1))</code> is not graphic.
1.3.20–1.3.21, 1.3.30	<code>cycles(K5, length=5)</code> , <code>cycles("K(3,4)", length=6)</code> , <code>cycles(petersen, length=5)</code> .
1.3.36–1.3.39	<code>deck(G)</code> .
1.3.45–1.3.49	<code>bipartite(G, sub=true)</code> .
1.3.52–1.3.54	<code>graph("turán(n,2)")</code> and <code>cycles(G, length=3)</code> (triangle-free?).
1.4.6, 1.4.26– 1.4.28	<code>euler("debruijn(2,3)")</code> ; remove words with <code>deledge</code> .
1.4.10–1.4.14	<code>components(D)</code> (strong components, acyclic order).
1.4.19–1.4.23	<code>euler(D)</code> for digraphs; <code>orient(G)</code> .
1.4.31, 1.4.34	<code>isomorphic</code> on tournaments; <code>degrees</code> gives the score sequence.
1.4.32	<code>graphic(p=(...), q=(...))</code> .
1.4.36–1.4.42	<code>kings("tournament(n, seed)")</code> : kings, the algorithm, a spanning path.

How the results are checked

The module's tests check the families (orders, sizes, degrees), the automorphism group orders of twenty graphs (Petersen 120, Q_4 384, Heawood 336, Grötzsch 10, ...), the numbers of graphs on up to six vertices (OEIS A000088 and A001349), every Eulerian circuit edge by edge, every de Bruijn sequence, Havel–Hakimi against Erdős–Gallai on 300 random lists, the large bipartite subgraph against the bound $m/2$, and kings against Prop. 1.4.30. They also check the facts of the exercises that the commands reproduce: the cycle counts of Exercises 1.1.25, 1.3.20, 1.3.21 and 1.3.30, the lists of Exercise 1.3.8, decompositions copy by copy, the five trails of Exercise 1.2.9, the orientations of Exercise 1.4.23, the king algorithm of Exercise 1.4.37, bigraphic lists against the Gale–Ryser inequalities, and the reconstruction of the kite (Exercise 1.3.36). When NetworkX is installed, they also compare isomorphism, cut-vertices, cut-edges, components, girth, strong components and graphic sequences with NetworkX on random graphs, and check that NetworkX reads the files that `graphsave` writes and that the module reads NetworkX's files.

Lab4IQM Studio and its modules are free software (GPL-3.0-or-later). This companion refers to West's book by section, theorem and exercise number; it reproduces none of its text or solutions. The numbers are those of the second edition.