RandomGraphExperiments_ForStudents_TOSEND

August 27, 2026

[2]:
```
from utils import empirical_means, empirical_dists
```

0.1 Introduction

In this series of exercises we will use the toolkit we've developed so far to study *random graphs*. As our mathematical model for "random graphs" we will take the probability space of labelled graphs on n vertices in which every edge is selected with probability $p \in [0,1]$.

To be more precise, we fix the vertex set to be $V = \{0, 1, 2, \dots, n-1\}$. As for the edge set E, we fix some $p \in [0,1]$ and we set $P(e \in E) = p$ for each of the $\binom{n}{2}$ possible edges between two vertices in V. To put it otherwise, the indicator function of u and v being neighbours (i.e. of the event $e = uv \in E$) is $X_{uv} \sim \text{Bernoulli}(p)$.

Therefore the probability of drawing a given $G = (V, E)$ is $P(G(n,p) = G) = p^{|E|}(1-p)^{\binom{n}{2}-|E|}$

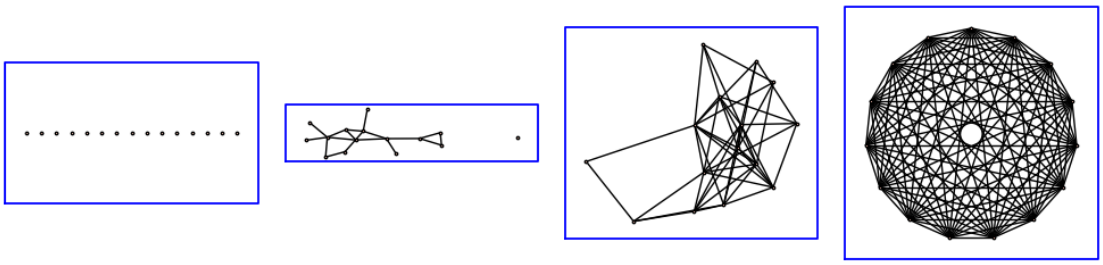
The exercises below will guide us in an exploration of how the structure of $G(n,p)$ evolves as a function of p. More precisely, we'll be interested in the asymptotic structure of $G(n,p)$, meaning we'll imagine $n \to \infty$.

To help us build some intuition, we'll carry Monte-Carlo experiments: we'll sample graphs out of $G(n,p)$ for varying values of p (and n big enough) and study various random variables on those graphs. This will help us to better visualise the content of the exercises, to formulate conjectures for their solution, and to help us verify our results.

Let's start by visualising $G(n,p)$ for small n and some fixed values of p. What do you observe?

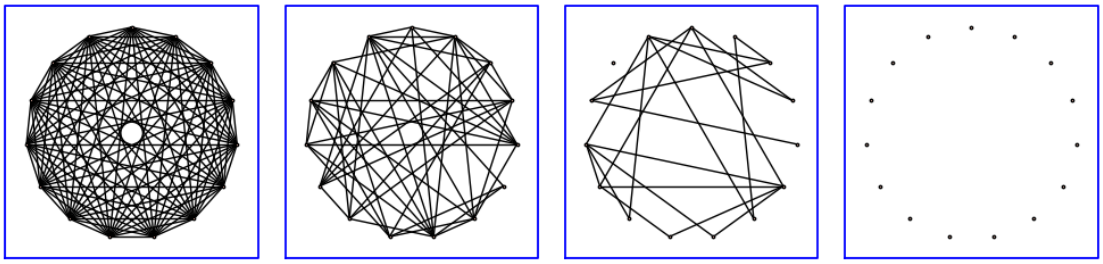
[31]:
```
n = 15

to_plot = []
for p in [0,0.11,1/2,1]:
    G = graphs.RandomGNP(n,p)
    to_plot.append(G.plot(vertex_size=3, vertex_labels=False,graph_border=True))

graphics_array(to_plot).show(figsize=[10,10])
```

```
The history saving thread hit an unexpected error (OperationalError('attempt to
write a readonly database')).History will not be written to the database.
```



Another way to think of $G(n,p)$ is as the so *Bernoulli bond percolation* model on the complete graph: starting with K_n, we delete edges with probability $1-p$.

[3]:
```
n = 15

to_plot = []
for p in [0,0.11,1/2,1]:
    G = graphs.RandomGNP(n,p)
    to_plot.append(G.plot(vertex_size=3,␣
 ↪vertex_labels=False,graph_border=True,layout='circular'))

graphics_array(to_plot[::-1]).show(figsize=[10,10])
```



0.2 Exercise 0: Degree distribution

It is clear that for $p=0$ all vertices have degree 0 and for $p=1$ they all have degree $n-1$. As a warmup, answer the following questions:

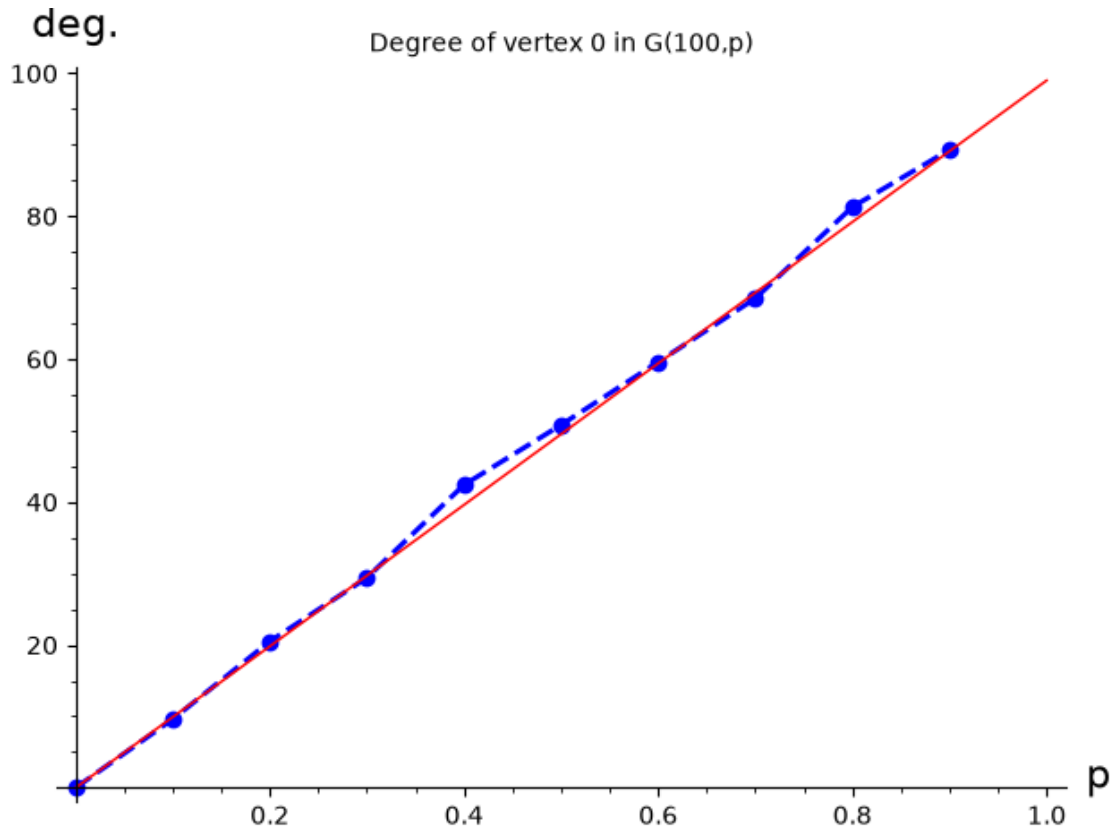
1. What is the average degree of a fixed vertex, say vertex 0, as a function of p?
2. What is the distribution of degrees for the same fixed vertex, as a function of p?

To help us in answering the above question, let's run our first Monte-Carlo experiments to calculate the empirical means and distributions as we vary p.

[46]:
```
c_range = [n/10 for n in range(0,10)] #breaking up the interval [0,1]
K = 10 #how many graphs to sample for each c
n = 100 #size of graphs to sample
```

```
#our random variable of interest: the degree of vertex 0
def degree_of_v(G):
    return G.degree(1)

avg_degree = empirical_means(n,c_range,degree_of_v,K,verbose=False,scale=lambda␣
 ↪n,c: c)
line(avg_degree,title="Degree of vertex 0 in G("+str(n)+",p)", \
     axes_labels=['p', 'deg.'],thickness=2,linestyle='--',marker='o') \
    + plot((n-1)*x, (x,0,1),color="red")
```

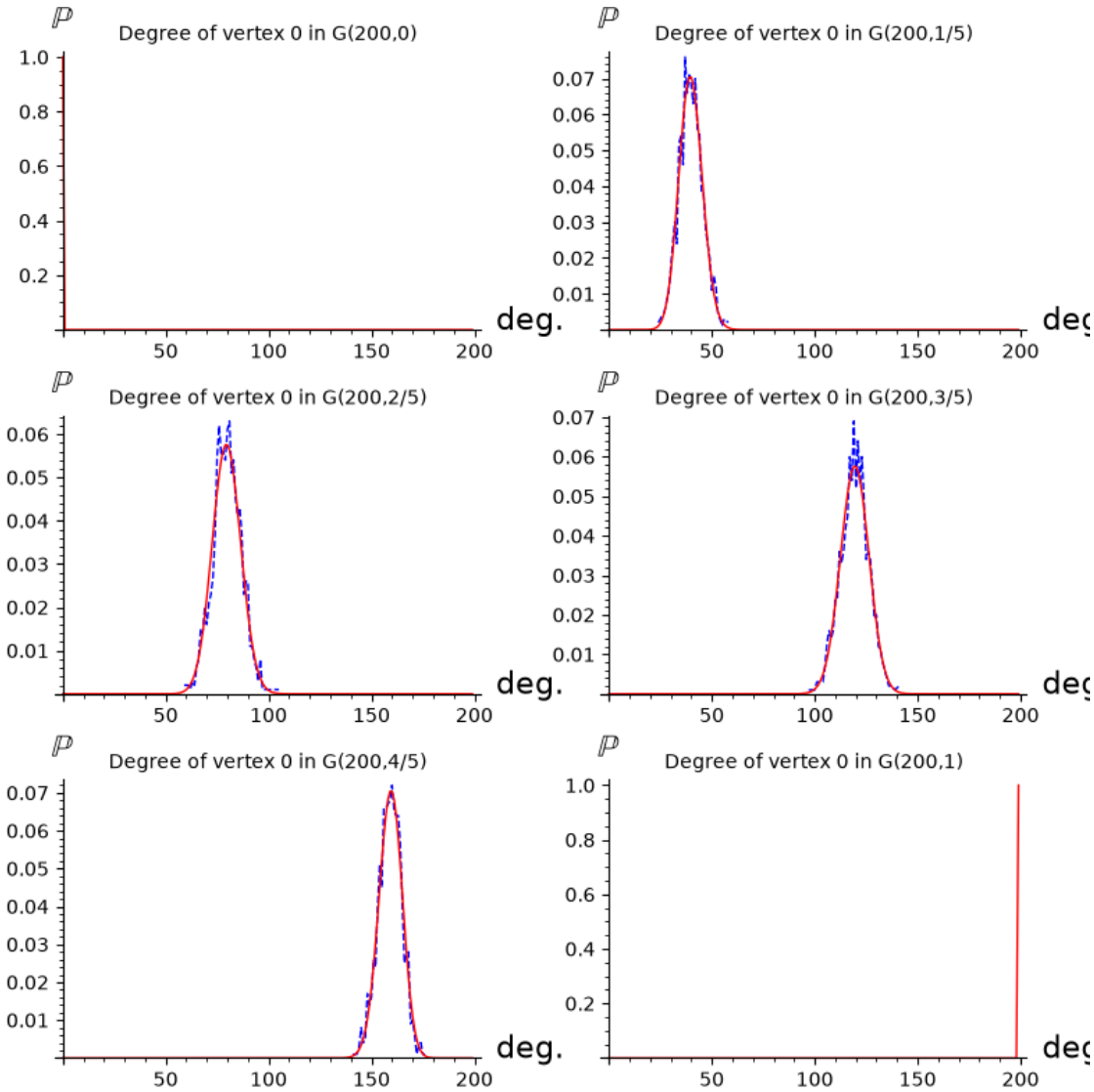
[46]:



[51]:

```
c_range = [c/10 for c in range(0,12,2)] #breaking up the interval [0,1]
K = 1000 #how many graphs to sample for each c
n = 200 #size of graphs to sample

degree_dists =␣
 ↪empirical_dists(n,c_range,degree_of_v,K,verbose=False,scale=lambda n,c: c)

def bernoulliPmf(n,p,k):
    return binomial(n-1,k) * p^k * (1-p)^(n-1-k)
```

```
to_plot = []
for c, data in degree_dists:
    m = len(data)
    to_plot.append(line2d(data,title="Degree of vertex 0 in␣
 ↪G("+str(n)+","+str(c)+")", \
                          axes_labels=['deg.',␣
 ↪'$\\mathbb{P}$'],thickness=1,linestyle='--') \
                          + line2d([(k,bernoulliPmf(n,c,k)) for k in␣
 ↪range(0,n)],color="red"))

graphics_array(to_plot,nrows=3).show(figsize=[8,8])
```



As we can see above, the graph starts as a bunch of isolated vertices (at $p = 0$) and eventually ends up as a complete graph (at $p = 1$). Of course this doesn't happen instantaneously: over the course, the graph develops a lot of interesting features, which will be our object of study.

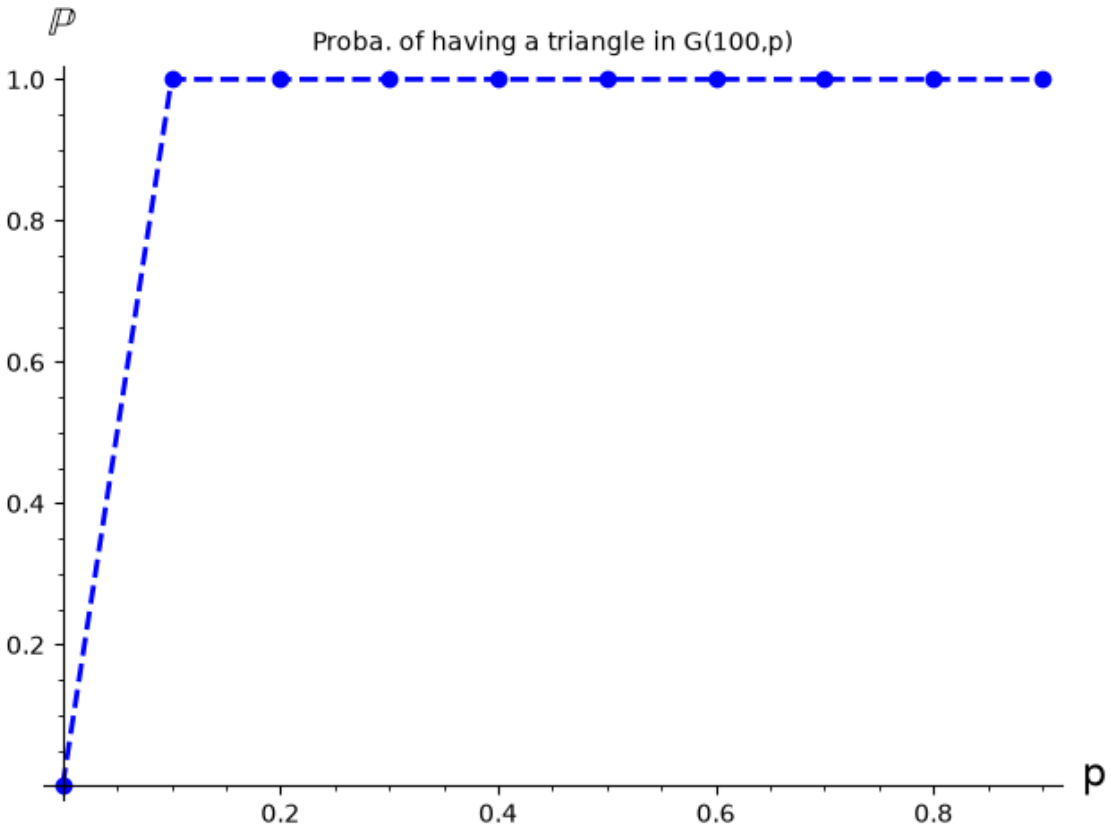
To get a good view of the emergence of those interesting structures, we'll often have to zoom in: having p vary over the entire range of $[0, 1]$ might lead us to miss a lot. Instead, we will have to figure out what range of values permits us to get a good glimpse.

To demonstrate this point, let's say we want to study the point at which triangles emerge (we'll come back to this problem later on!). For our first experiment, we'll let p take values over the entire range $[0, 1]$.

Note: as an empirical estimate of a probability, we'll often take the empirical mean of the corresponding indicator variable, here $1_{\text{num. triangles}>0}$.

[54]:
```
c_range = [n/10 for n in range(0,10)] #breaking up the interval [0,1]
K = 10 #how many graphs to sample for each c
n = 100 #size of graphs to sample

#a function which computes our statistic of interest: the presence of triangles
def has_triangle(G):
    if G.triangles_count() > 0:
        return 1
    else:
        return 0

has_tri = empirical_means(n,c_range,has_triangle,K,verbose=False,scale=lambda␣
 ↪n,c: c)
line(has_tri,title="Proba. of having a triangle in G("+str(n)+",p)", \
     axes_labels=['p', '$\\mathbb{P}$'],thickness=2,linestyle='--',marker='o')
```

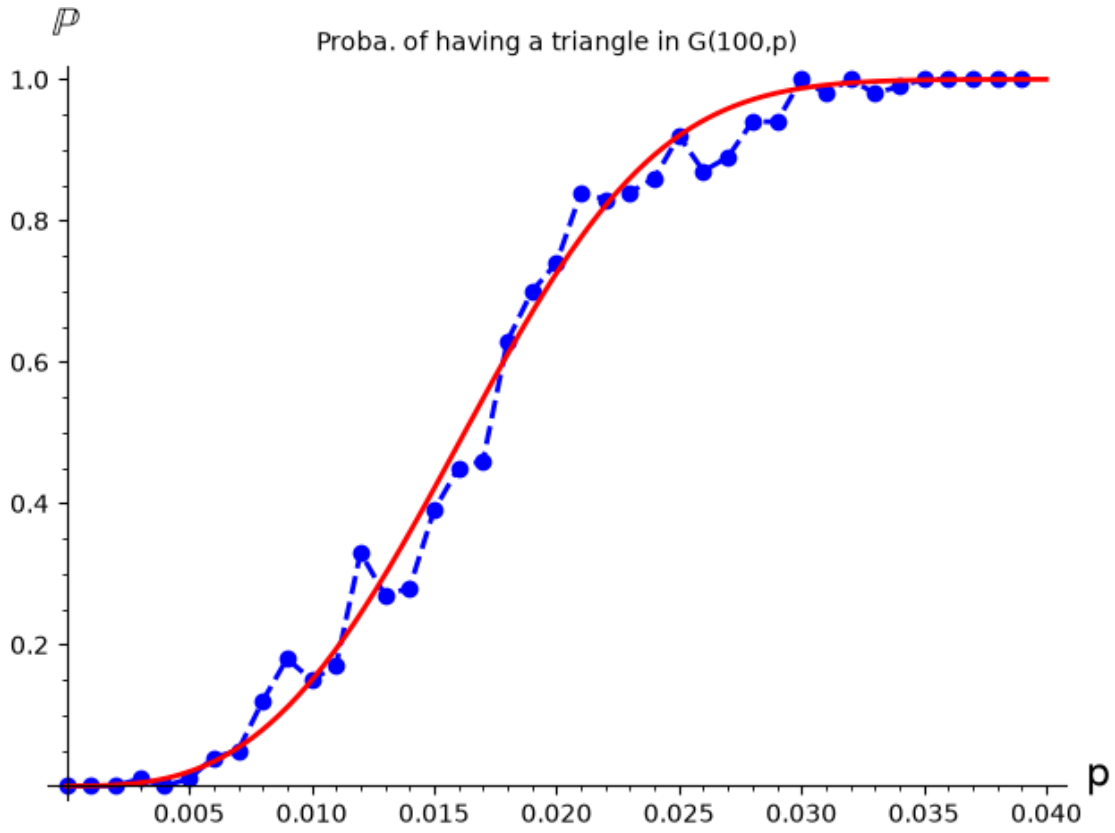
[54]:



As we can see from the plot above, the (empirical) probability of $G(n,p)$ to have a triangle seems to jump from 0 directly to 1 at some point. Does the emergence of triangles really happen in such a discontinuous way as the picture suggests?

The answer is no! But to see this, we must zoom into the range of values for p around which the transition from 0 to 1 takes place. This is around $O(n^{-2})$, as we'll later see. Indeed, our techniques not only predict the proper range into which we must zoom into but they even give us analytic results (the red curve) on the precise shape of this transition!

[55]:
```
c_range = [n/1000 for n in range(0,40)] #breaking up the interval [1,3]
K = 100 #how many graphs to sample for each c
n = 100 #size of graphs to sample

has_tri = empirical_means(n,c_range,has_triangle,K,verbose=False,scale=lambda␣
 ↪n,c: c)
line(has_tri,title="Proba. of having a triangle in G("+str(n)+",p)", \
     axes_labels=['p', '$\\mathbb{P}$'],thickness=2,linestyle='--',marker='o') \
     + plot(1-exp(-binomial(100,3)*x^3), (x,0,0.04),color="red",thickness=2)
```
[55]:



Our exploration begins with the study of the occurences of fixed patterns of interest.

For a fixed pattern such as triangles, our plot above leads us to conjecture the existence of three regions: one in which the the probability of occurence is 0, another in which it is 1, and a transition between the two.

Work by Bollobas and Thomassen shows that for all *monotone* (= properties that are preserved by the addition of edges) properties, this is indeed the case. We will always chop $[0, 1]$ into a three parts: - Subcritical regime: the probability that the structure occurs tends to 0. - Critical regime: the probability has non-trivial limiting behaviour. - Supercritical regime: the probability that the structure occurs tends to 1.

To aid you in approaching the questions below, the following recipe is often applicable: - Define X_n counting occurences of the pattern of interest. - Compute $\mathbb{E}(X_n)$ and use Markov's inequality to find the subcritical phase. - Compute $\mathrm{Var}(X_n)$ and use Chebyshev's inequality to find the supercritical phase. - Use the method of moments to figure out the distribution inside the critical window.

0.3 Exercise 1: Edges in $G(n, p)$

In this exercise, you study the occurence of edges, by answering the following: 1. What's the threshold function for the presence of an edge in a random graph?
Hint: In this particularly simple case, you can directly compute $\mathbb{P}(X = k)$.
Of course, the first moment/second moment recipe still applies. 2. What's the limiting distribution

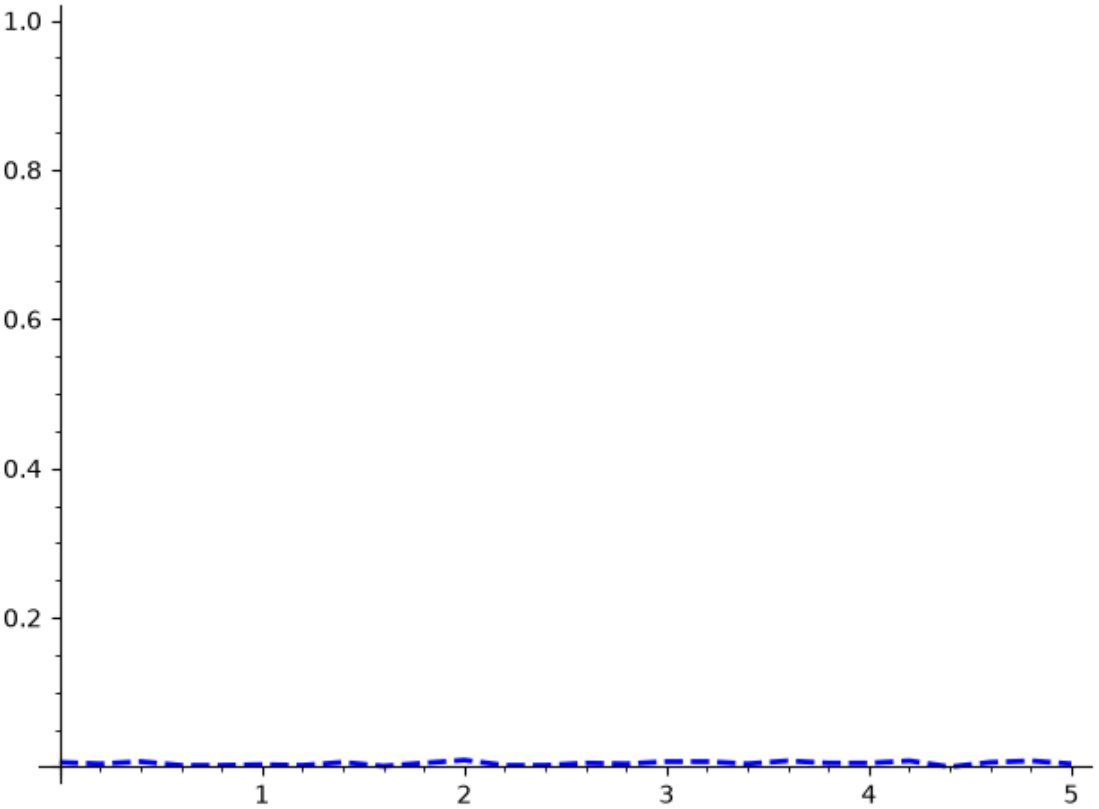
for the number of edges?
Hint: Again, $\mathbb{P}(X = k)$ should look familiar! Consult the notes for the relevant limit theorem.
If you prefer to follow the recipe, you can always attempt the method of moments. 3. What's the limiting distribution for the indicator variable corresponding to the *existence of at least one edge*?
Hint: You can either work this out directly using $\mathbb{P}(X = k)$ or by working with the limit distribution for the number of edges.

For $c = o(n^{-2})$ we have no edges, w.h.p.

[6]:
```
c_range = [x / 10.0 for x in range(0, 51, 2)] #breaking up the interval [1,3]
K = 1000 #how many graphs to sample for each c
n = 100 #size of graphs to sample

def has_edge(G):
    if G.num_edges() > 0:
        return 1
    else:
        return 0

proba_edge = empirical_means(n,c_range,has_edge,K,verbose=False,scale=lambda␣
↪n,c: 1/n^3)
```

[7]:
```
line(proba_edge,thickness=2,linestyle='--',ymax=1)
```

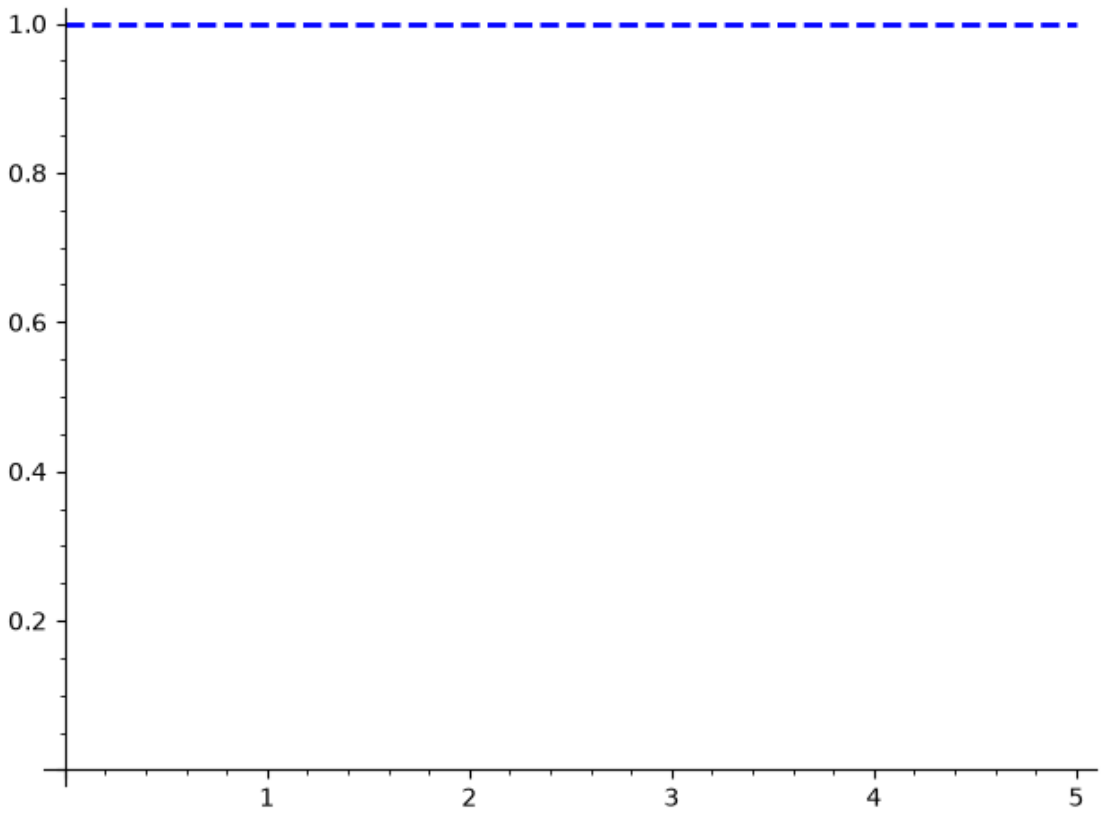
[7]:



For $c = \omega(n^{-2})$ we have at least one edge, w.h.p.

[8]:
```
proba_edge = empirical_means(n,c_range,has_edge,K,verbose=False,scale=lambda␣
↪n,c: 1/n)
line(proba_edge,thickness=2,linestyle='--',ymax=1)
```

[8]:

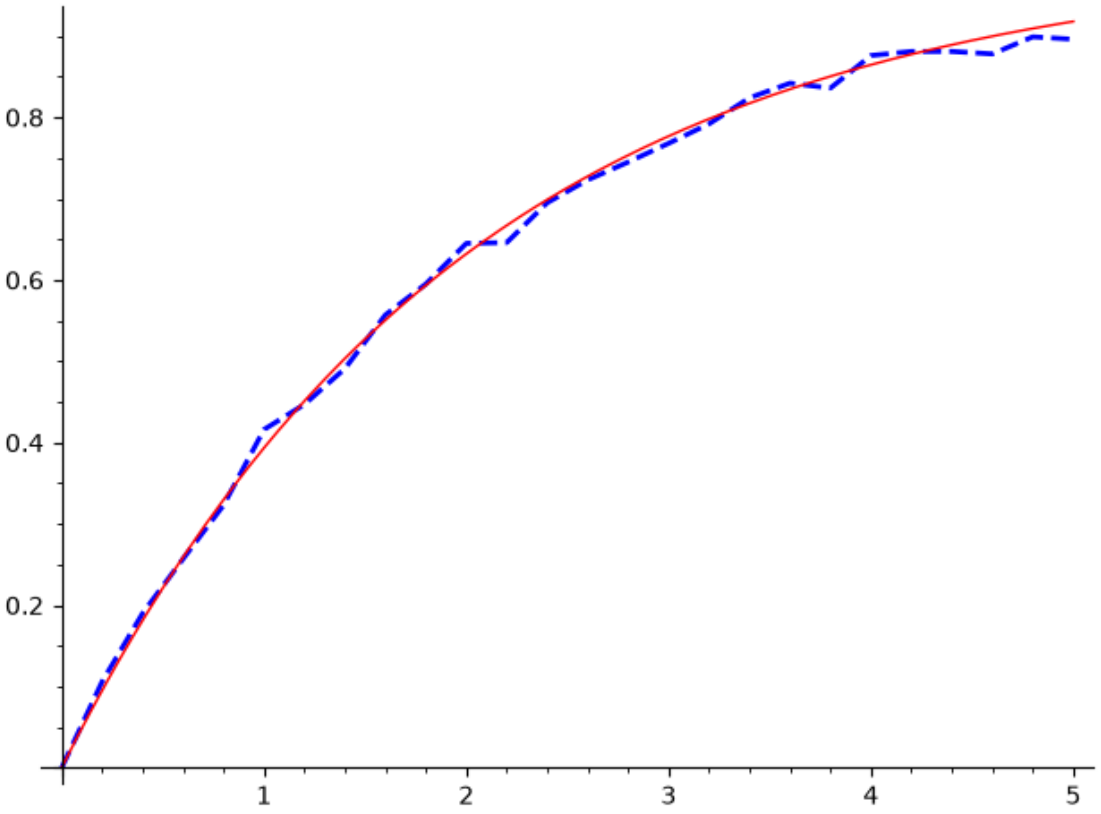


Inside the window $c = O(n^{-2})$ we observe the evolution of the probabilities for an edge to exist.

[9]:
```
c_range = [x / 10.0 for x in range(0, 51, 2)] #breaking up the interval [1,3]
K = 1000 #how many graphs to sample for each c
n = 100 #size of graphs to sample

proba_edge = empirical_means(n,c_range,has_edge,K,verbose=False,scale=lambda␣
↪n,c: c/n^2)
```

[10]:
```
line(proba_edge,thickness=2,linestyle='--') + plot(1-exp(-x/2),␣
↪(x,0,5),color="red")
```

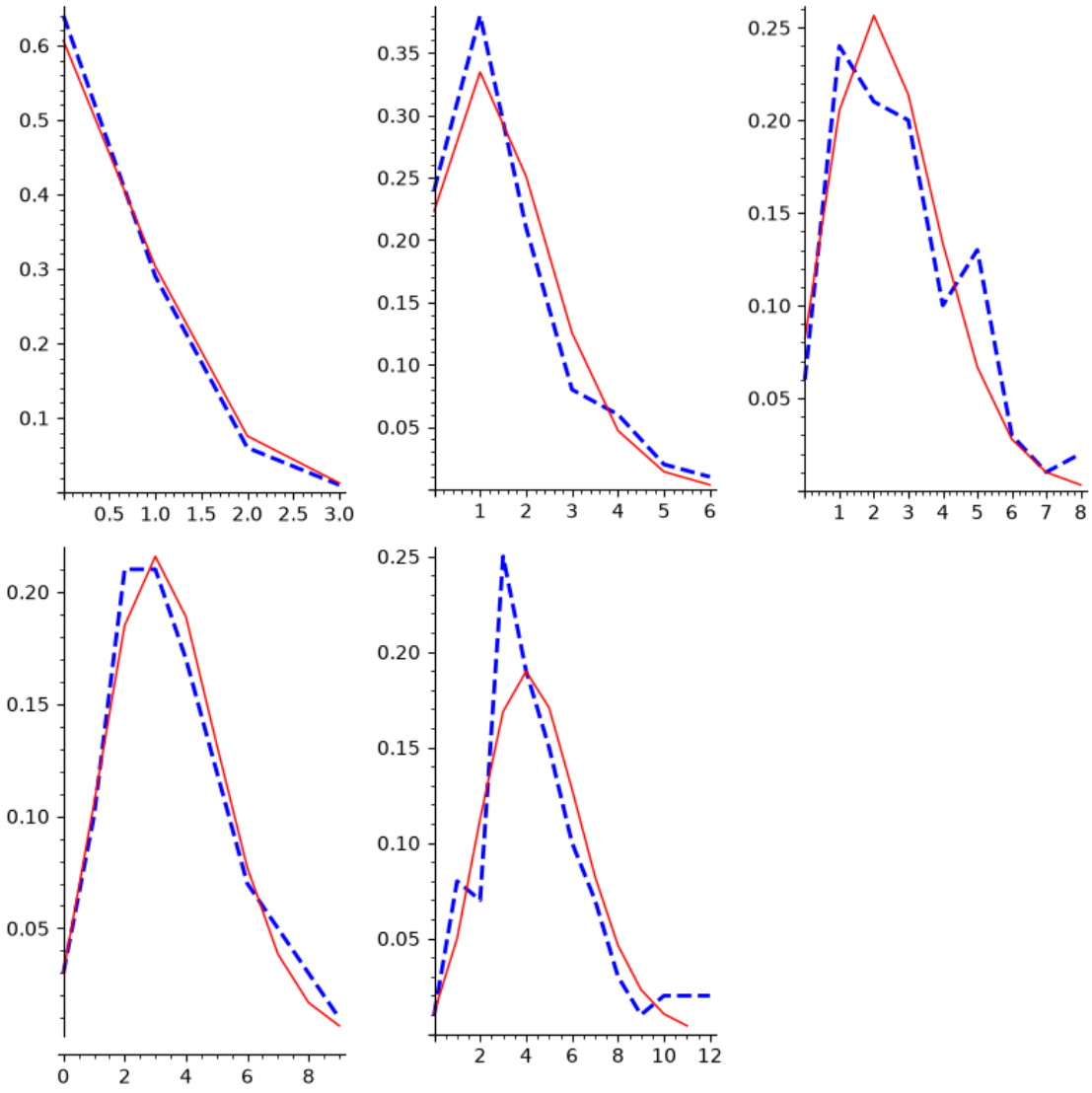
[10]:



Let's now plot the empirical distributions for the *number of* edges in $G(n, p = cn^{-2})$ as we vary the constant c.

[64]:
```
c_range = [c for c in range(1,10,2)] #breaking up the interval [0,1]
K = 100 #how many graphs to sample for each c
n = 1000 #size of graphs to sample

res = empirical_dists(n,c_range,lambda g: g.num_edges(),K,scale=lambda n,c:
 ↪c*n^(-2),verbose=True)

def poisPmf(c,k):
    return (c/2)^k/factorial(k) * exp(-c/2).n()

to_plot = []

for c, data in res:
    m = len(data)
    to_plot.append(line2d(data,thickness=2,linestyle='--') +␣
 ↪line2d([(k,poisPmf(c,k)) for k in range(0,m)],color="red"))

graphics_array(to_plot,nrows=2).show(figsize=[8,8])
```

```
Currently sampling with c = 1
COUNTS IS Counter({0: 64, 1: 29, 2: 6, 3: 1})
Currently sampling with c = 3
COUNTS IS Counter({1: 38, 0: 24, 2: 21, 3: 8, 4: 6, 5: 2, 6: 1})
Currently sampling with c = 5
COUNTS IS Counter({1: 24, 2: 21, 3: 20, 5: 13, 4: 10, 0: 6, 6: 3, 8: 2, 7: 1})
Currently sampling with c = 7
COUNTS IS Counter({2: 21, 3: 21, 4: 17, 5: 12, 1: 10, 6: 7, 7: 5, 8: 3, 0: 3, 9:
1})
Currently sampling with c = 9
COUNTS IS Counter({3: 25, 4: 19, 5: 15, 6: 10, 1: 8, 7: 7, 2: 7, 8: 3, 12: 2,
10: 2, 9: 1, 0: 1})
```



0.4 Exercise 2: 3-stars and 4-paths in $G(n,p)$

0.4.1 Main ideas: induced vs not-induced subgraphs, monotonicity, automorphisms

As an extension let us ask the same questions as above but this time for the 3-star and the 4-path, the graphs depicted below. In more detail, we ask:

1. What's the threshold function for the presence of a 3-star/4-path in a random graph?
2. What's the limiting distribution for the number of 3-stars/4-path?

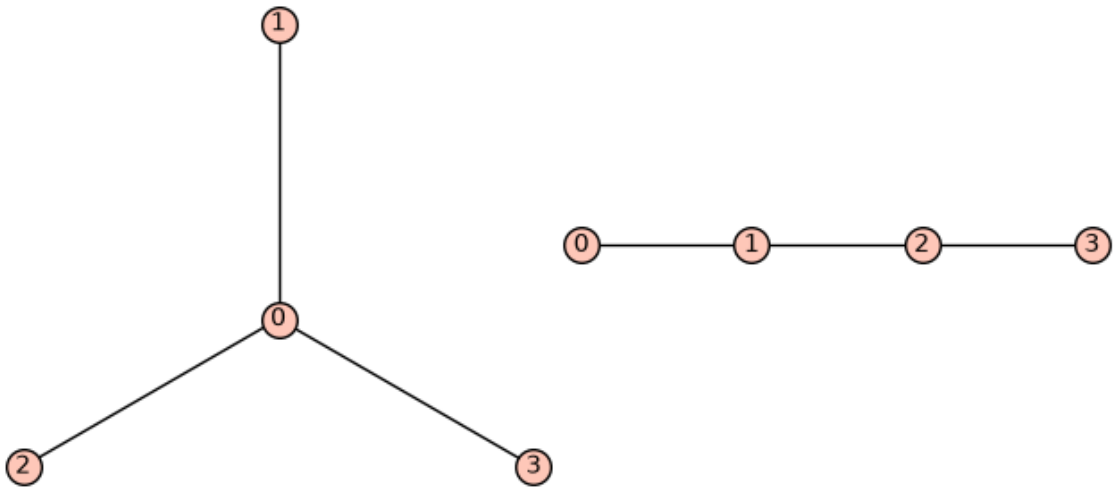
3. What's the limiting distribution for the indicator variable corresponding to the *existence of at least one 3-star/4-path*?

Before delving into the questions above, we have an important caveat to discuss here: what do we mean by "occurence of a pattern"?
Earlier we discussed the occurence of edges, where it was clear what we meant. For more general graphs (such as the 3-star and the 4-path) however, the intuitive idea of occurence is ambiguous: are we studying fixed graphs as subgraph or specifically as an *induced* subgraphs of $G(n,p)$? Furthermore, do we count *labelled* copies or *unlabelled* ones?

Concerning the first question, the notion of thresholds we introduced earlier helps us narrow it down: only in the not-necessarily-induced case does it makes sense to search for thresholds.

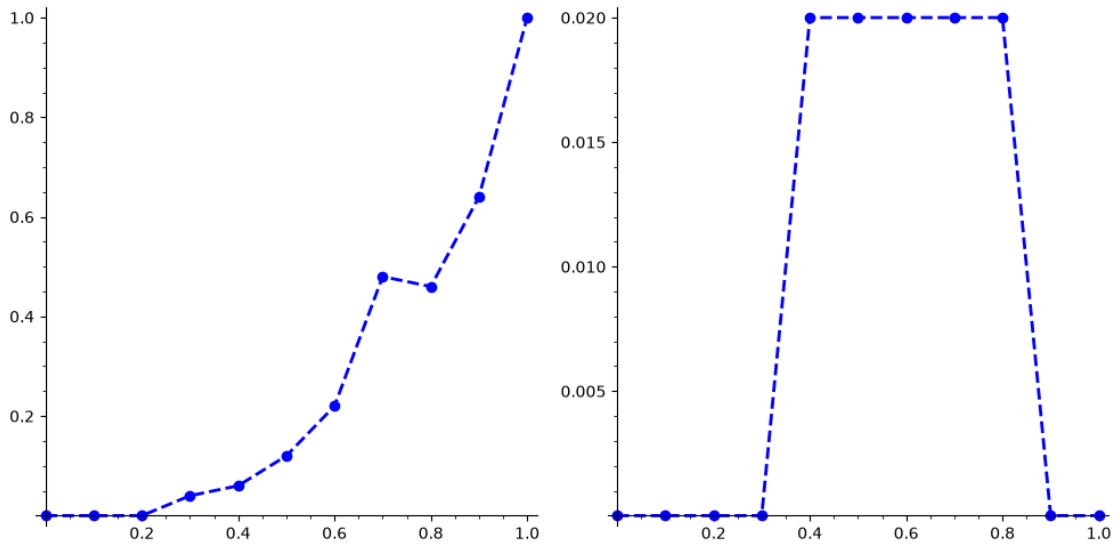
To see why this is so, let's plot the empirical probability of having a 3-star as a subgraph and as an induced subgraph. What do we notice in terms of monotonicity of the resulting graphs?

[101]:
```
graphics_array([graphs.StarGraph(3).plot(figsize=[3,3]), graphs.PathGraph(4).
↪plot(figsize=[3,3])]).show()
```



[102]:
```
c_range = [n/10 for n in range(0,11)] #breaking up the interval [0,1]
K = 50 #how many graphs to sample for each c
n = 100 #size of graphs to sample
```

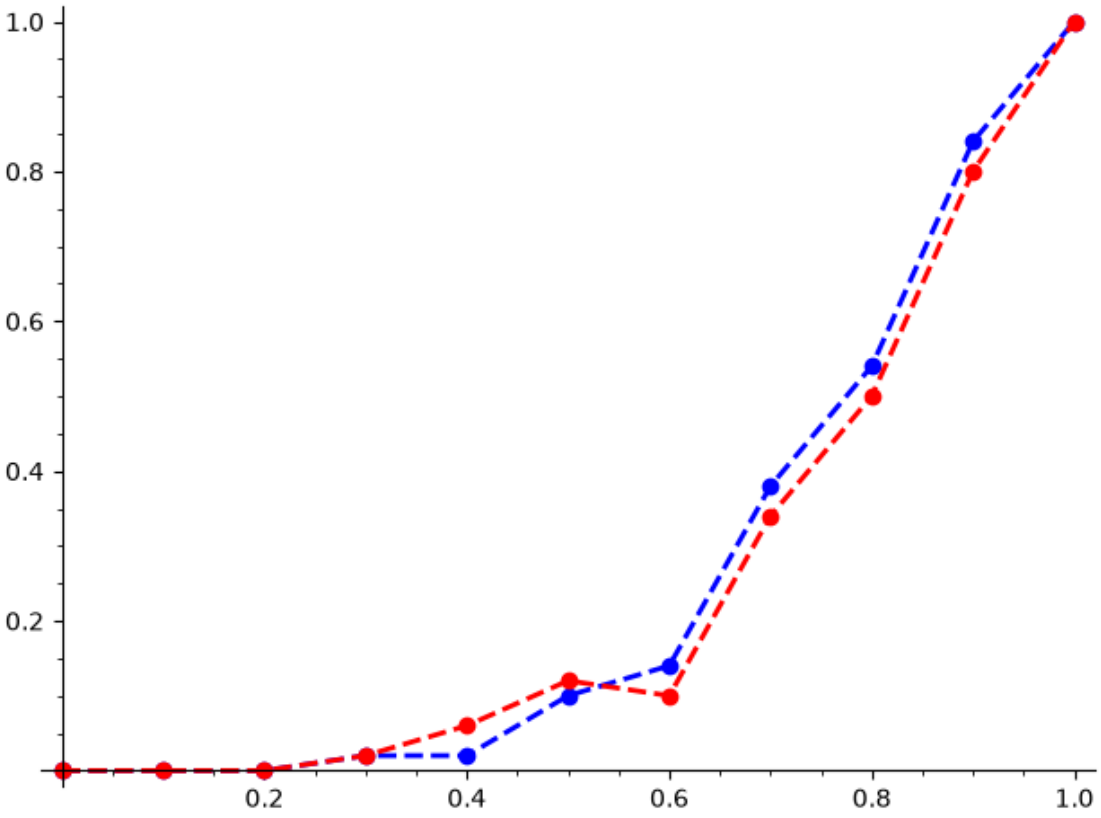
```
def has_induced_star(G):

    T = graphs.StarGraph(3)
    if T.is_subgraph(G,induced=True):
        return 1
    else:
        return 0

def has_star(G):

    T = graphs.StarGraph(3)
    if T.is_subgraph(G,induced=False):
        return 1
    else:
        return 0

stars_noninduced =␣
 ↪empirical_means(n,c_range,has_star,K,verbose=False,scale=lambda n,c: c)
p1 = line(stars_noninduced,thickness=2,linestyle='--',marker='o')

stars_induced =␣
 ↪empirical_means(n,c_range,has_induced_star,K,verbose=False,scale=lambda n,c:␣
 ↪c)
p2 = line(stars_induced,thickness=2,linestyle='--',marker='o')

graphics_array([p1,p2],nrows=1).show(figsize=[10,5])
```



Let's now overlay the empirical probabilities of a 3-star and a 4-path existing. What do you notice? How do you explain this?

[114]:
```
c_range = [n/10 for n in range(0,11)] #breaking up the interval [0,1]
K = 50 #how many graphs to sample for each c
n = 100 #size of graphs to sample

def has_path(G):

    T = graphs.PathGraph(4)
    if T.is_subgraph(G,induced=False):
        return 1
    else:
        return 0

paths = empirical_means(n,c_range,has_path,K,verbose=False,scale=lambda n,c: c)
p3 = line(paths,thickness=2,linestyle='--',marker='o',color="red")

p1+p3
```

[114]:

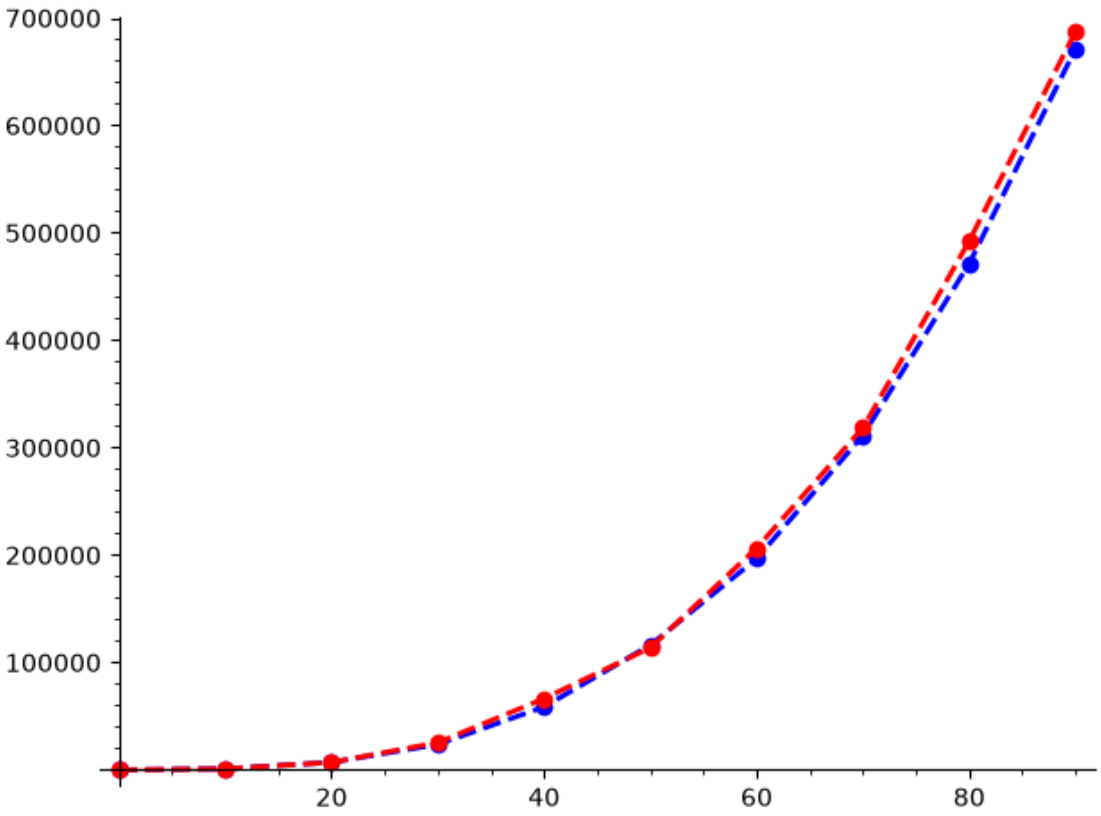


What about the mean number of these two subgraphs? How do you explain the results for the case of labelled and unlabelled occurences, as seen below?

[134]:
```
c_range = [n for n in range(0,100,10)] #breaking up the interval [0,1]
K = 10 #how many graphs to sample for each c
n = 100 #size of graphs to sample

def num_paths(G):
    T = graphs.PathGraph(4)
    return G.subgraph_search_count(T,induced=False) #the built-in method counts␣
 ↪*labelled* copies!

def num_stars(G):
    T = graphs.StarGraph(3)
    return G.subgraph_search_count(T,induced=False)

count_paths = empirical_means(n,c_range,num_paths,K,verbose=False,scale=lambda␣
 ↪n,c: c*n^(-4/3))
count_stars = empirical_means(n,c_range,num_stars,K,verbose=False,scale=lambda␣
 ↪n,c: c*n^(-4/3))

p4 = line(count_paths,thickness=2,linestyle='--',marker='o',color="blue")
p5 = line(count_stars,thickness=2,linestyle='--',marker='o',color="red")

p4+p5
```

```
Currently sampling with c = 0
Currently sampling with c = 10
Currently sampling with c = 20
Currently sampling with c = 30
Currently sampling with c = 40
Currently sampling with c = 50
Currently sampling with c = 60
Currently sampling with c = 70
Currently sampling with c = 80
Currently sampling with c = 90
Currently sampling with c = 0
Currently sampling with c = 10
Currently sampling with c = 20
Currently sampling with c = 30
Currently sampling with c = 40
Currently sampling with c = 50
Currently sampling with c = 60
Currently sampling with c = 70
Currently sampling with c = 80
Currently sampling with c = 90
```

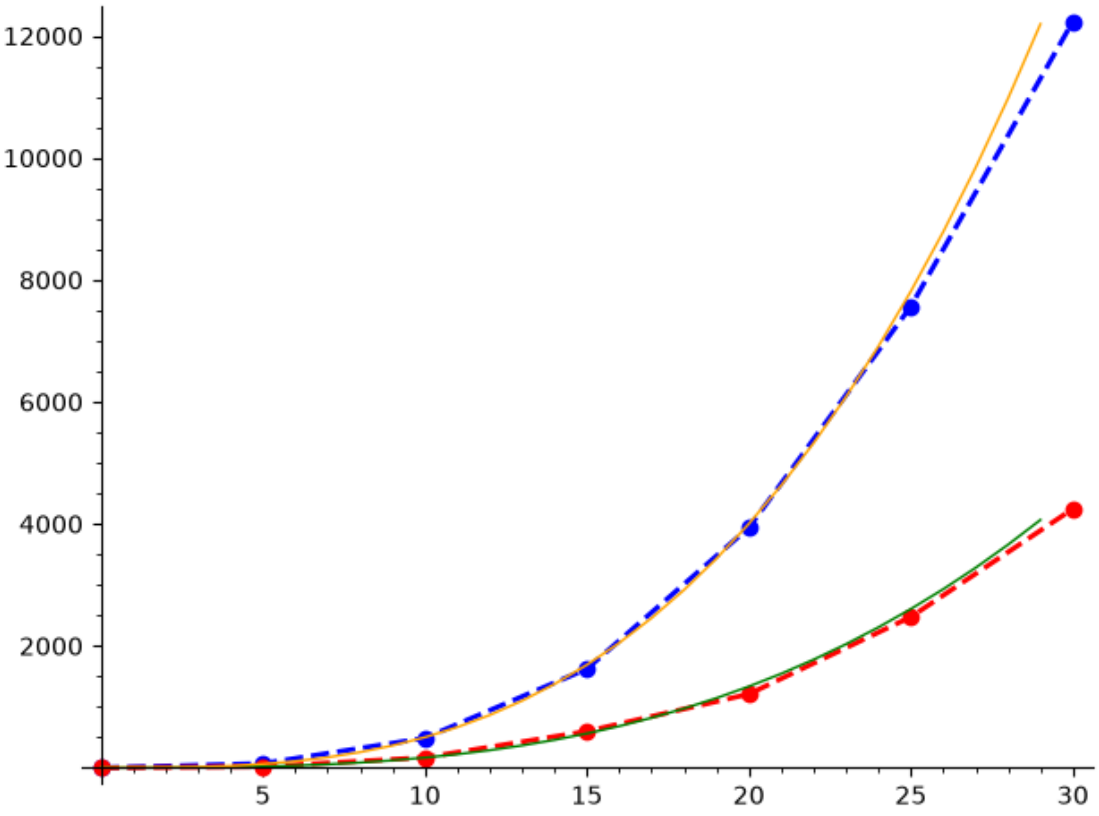
[134]:



[3]:
```
c_range = [n for n in range(0,35,5)] #breaking up the interval [0,1]
K = 10 #how many graphs to sample for each c
n = 200 #size of graphs to sample

def num_paths_unlabelled(G):
    T = graphs.PathGraph(4)
    return len({frozenset(emb) for emb in G.
 ↪subgraph_search_iterator(T,induced=False)})

def num_stars_unlabelled(G):
    T = graphs.StarGraph(3)
    return len({frozenset(emb) for emb in G.
 ↪subgraph_search_iterator(T,induced=False)})

count_paths =␣
 ↪empirical_means(n,c_range,num_paths_unlabelled,K,verbose=False,scale=lambda␣
 ↪n,c: c*n^(-4/3))
count_stars =␣
 ↪empirical_means(n,c_range,num_stars_unlabelled,K,verbose=False,scale=lambda␣
 ↪n,c: c*n^(-4/3))
```

```
p6 = line(count_paths,thickness=2,linestyle='--',marker='o',color="blue")
p7 = line(count_stars,thickness=2,linestyle='--',marker='o',color="red")

p6 + p7 + line2d([(x,x^3/6) for x in range(0,30)],color="green") +␣
 ↪line2d([(x,x^3/2) for x in range(0,30)],color="orange")
```

[3]:

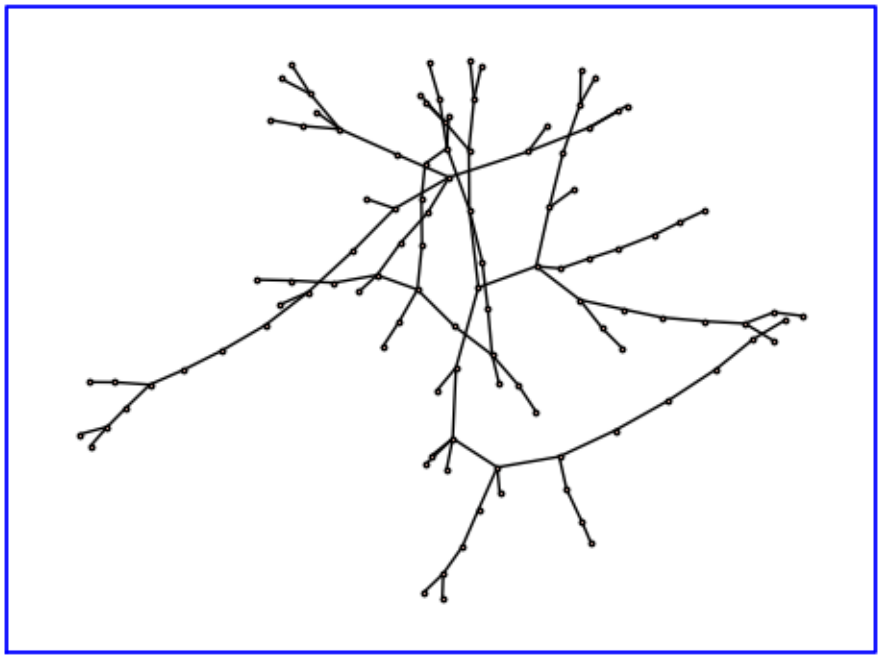


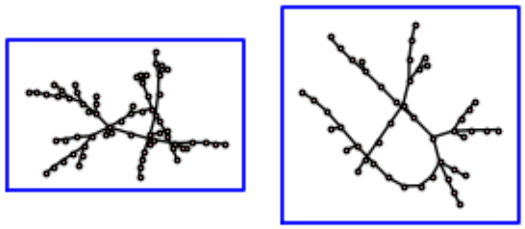
So far we've studied tree-like patterns, which lead to us mostly exploring the regime $p = o(n)$. Let us now turn up p, focusing on the sparse regime $p = O(n)$, i.e. let's look at $G(n, p = c/n)$. We'll begin by plotting the, say, three biggest components. What do you notice as you vary $c \in \mathbb{R}_{\geq 0}$?

[27]:
```
n = 1000
c = 1.1

G = graphs.RandomGNP(n,c/n)

ccs_by_size = sorted(G.connected_components_subgraphs(),key=lambda x: x.
 ↪size(),reverse=True)

to_plot = []
```

```
for cc in ccs_by_size[0:min(3,len(ccs_by_size))]:
    scale_by_size = (n*10)/cc.size()
    to_plot.append(cc.plot(vertex_size=3,␣
 ↪vertex_labels=False,graph_border=True))
```

[28]:
```
to_plot[0].show(figsize=[5,5]) #plot the big one first
if to_plot[1:] != []:
    graphics_array(to_plot[1:]).show(figsize=[3,3]) #then the two smaller ones,␣
 ↪if they even exist!
```





0.5 Exercise 3: Beyond trees

So far we've studied small tree-like subgraphs: edges, stars, paths.
Based on our simulations, we guess that $G(n, c/p)$ looks, at least locally, quite tree-like.

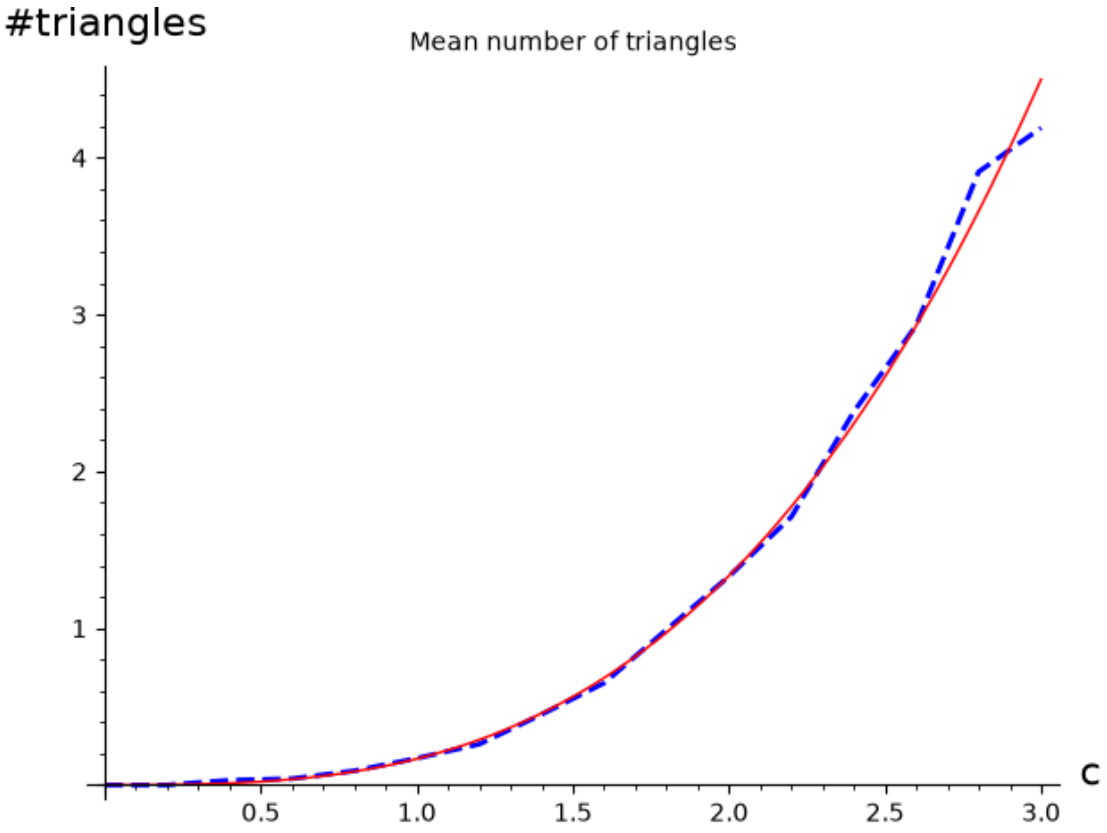
This exercise focuses on the appearance of the first small cycles, focusing on triangles. In more detail we ask: 1. What's the threshold function for the presence of a triangle in a random graph?
2. What's the limiting distribution for the number of triangles?
3. What's the limiting distribution for the indicator variable corresponding to the *existence of at least one triangle*?

[13]:
```
c_range = [x / 10.0 for x in range(0, 31, 2)] #breaking up the interval [1,3]
K = 100 #how many graphs to sample for each c
n = 500 #size of graphs to sample

num_tris = empirical_means(n,c_range,lambda G: G.
 ↪triangles_count(),K,verbose=False)
has_tris = empirical_means(n,c_range,lambda G: 1 if (G.triangles_count() > 0)␣
 ↪else 0,K,verbose=False)
```

```
Currently sampling with c = 0.000000000000000
Currently sampling with c = 0.200000000000000
Currently sampling with c = 0.400000000000000
Currently sampling with c = 0.600000000000000
Currently sampling with c = 0.800000000000000
Currently sampling with c = 1.00000000000000
Currently sampling with c = 1.20000000000000
Currently sampling with c = 1.40000000000000
Currently sampling with c = 1.60000000000000
Currently sampling with c = 1.80000000000000
Currently sampling with c = 2.00000000000000
Currently sampling with c = 2.20000000000000
Currently sampling with c = 2.40000000000000
Currently sampling with c = 2.60000000000000
Currently sampling with c = 2.80000000000000
Currently sampling with c = 3.00000000000000
```

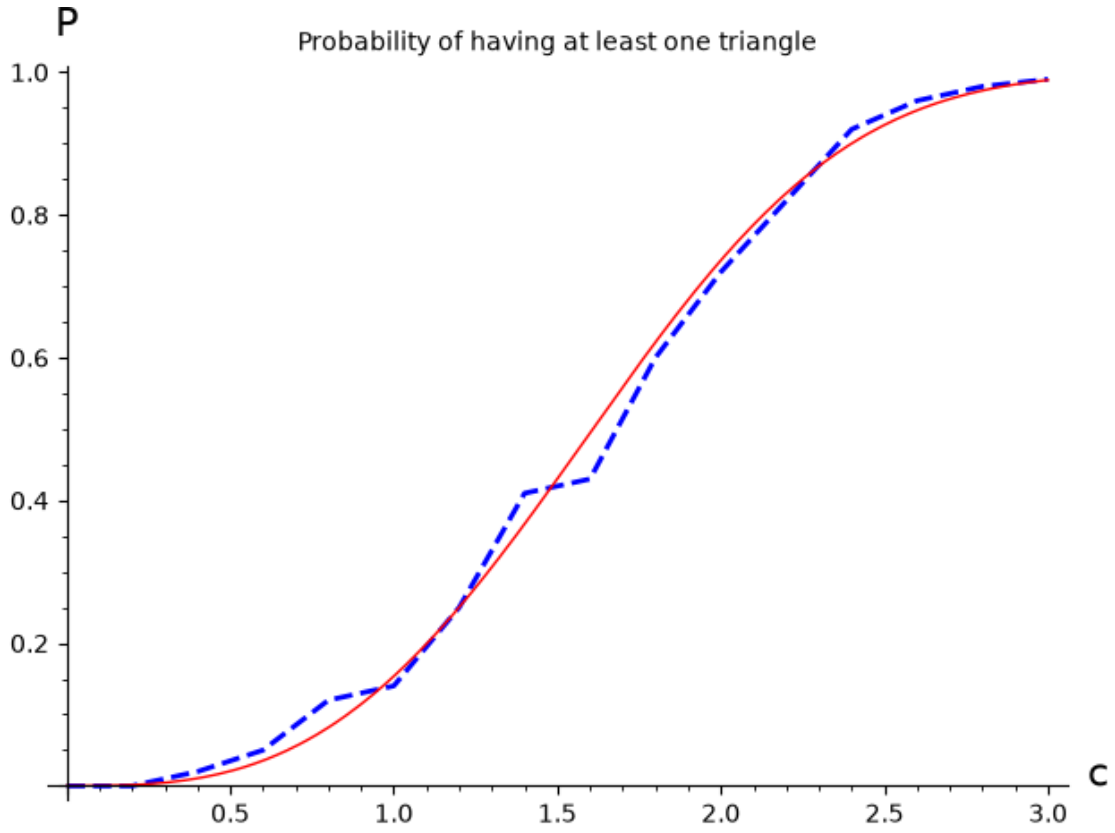
[14]:
```
line(num_tris,title="Mean number of triangles",axes_labels=['c',␣
 ↪'#triangles'],thickness=2,linestyle='--') \
+ plot(x^3/6 ,(x,0,3),color="red")
```

[14]:



[15]:
```
line(has_tris,title="Probability of having at least one triangle",axes_labels=['c', 'P'],thickness=2,linestyle='--') \
+ plot((1-exp(-x^3/6)),(x,0,3),color="red")
```

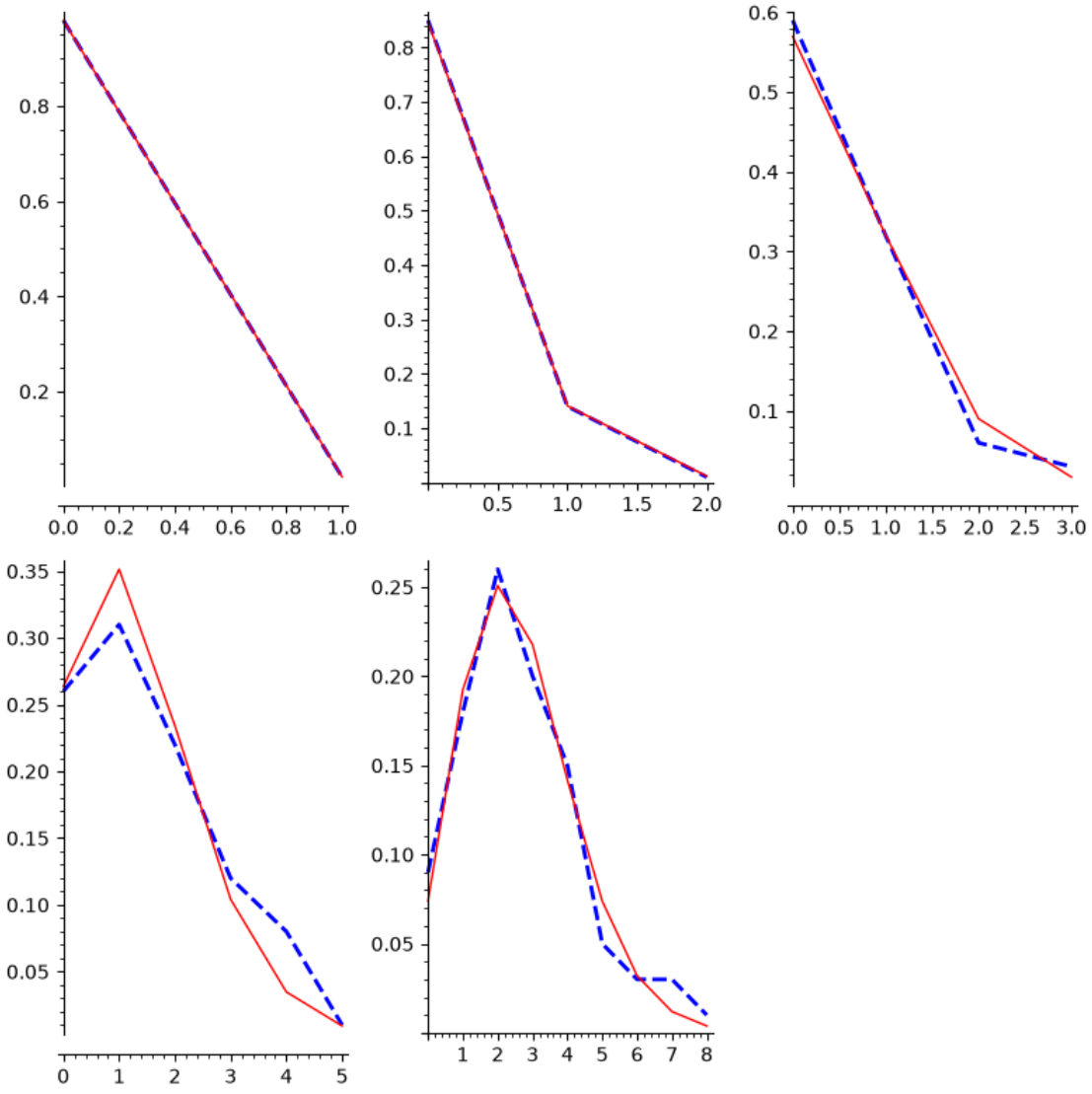
[15]:



[10]:
```
c_range = [c/10 for c in range(5,30,5)] #breaking up the interval [1/2,3]
K = 100 #how many graphs to sample for each c
n = 1000 #size of graphs to sample

res = empirical_dists(n,c_range,lambda g: g.triangles_count(),K,scale=lambda␣
 ↪n,c:c/n,verbose=True)

def poisPmf(c,k):
    return (c^3/6)^k/factorial(k) * exp(-c^3/6).n()


to_plot = []

for c, data in res:
    m = len(data)
    to_plot.append(line2d(data,thickness=2,linestyle='--') +␣
 ↪line2d([(k,poisPmf(c,k)) for k in range(0,m)],color="red"))

graphics_array(to_plot,nrows=2).show(figsize=[8,8])
```

```
Currently sampling with c = 1/2
```

```
Data points: Counter({0: 98, 1: 2})
Currently sampling with c = 1
Data points: Counter({0: 85, 1: 14, 2: 1})
Currently sampling with c = 3/2
Data points: Counter({0: 59, 1: 32, 2: 6, 3: 3})
Currently sampling with c = 2
Data points: Counter({1: 31, 0: 26, 2: 22, 3: 12, 4: 8, 5: 1})
Currently sampling with c = 5/2
Data points: Counter({2: 26, 3: 20, 1: 18, 4: 15, 0: 9, 5: 5, 7: 3, 6: 3, 8: 1})
```

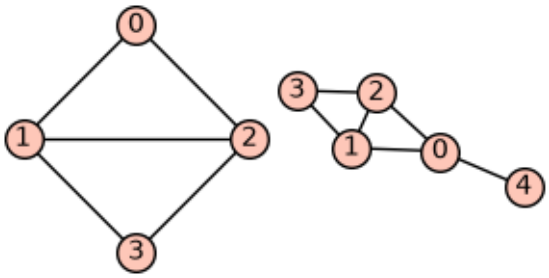


0.6 Exercise 4: Diamonds and spatulas

The diamond and spatula graphs are the two graphs depicted below. Let's try to find their threshold functions.

We start with the diamond graph: 1. What is the threshold function for the existence of at least one diamond?

[26]:
```
diam = graphs.DiamondGraph()
spat = graphs.DiamondGraph() #the spatula is the diamond plus a new edge, so I␣
 ↪start with a diamond

spat.add_vertex(4)
spat.add_edge((0,4))

graphics_array([diam.plot(), spat.plot()],nrows=1).show(figsize=[3,3])
```



You might have noticed that so far that the first-moment calculations were enough for us to guess the correct threshold. This was certainly the case for the diamond graph above.

Below we plot the probability of having at least one spatula, as well as the mean number of spatulas for $p = O(n^{-5/6})$, the potential threshold suggested by the first-moment method.

1. What do you observe?
2. How do you explain this?
3. What is the true threshold? How is it related to the threshold you calculated for the diamond?

Hint: Think back to the example of X with $\mathbb{E}(X) \to +\infty$ but $\mathbb{P}(X) \to 0$ of the notes. What was the intuitive cause of this phenomenon?

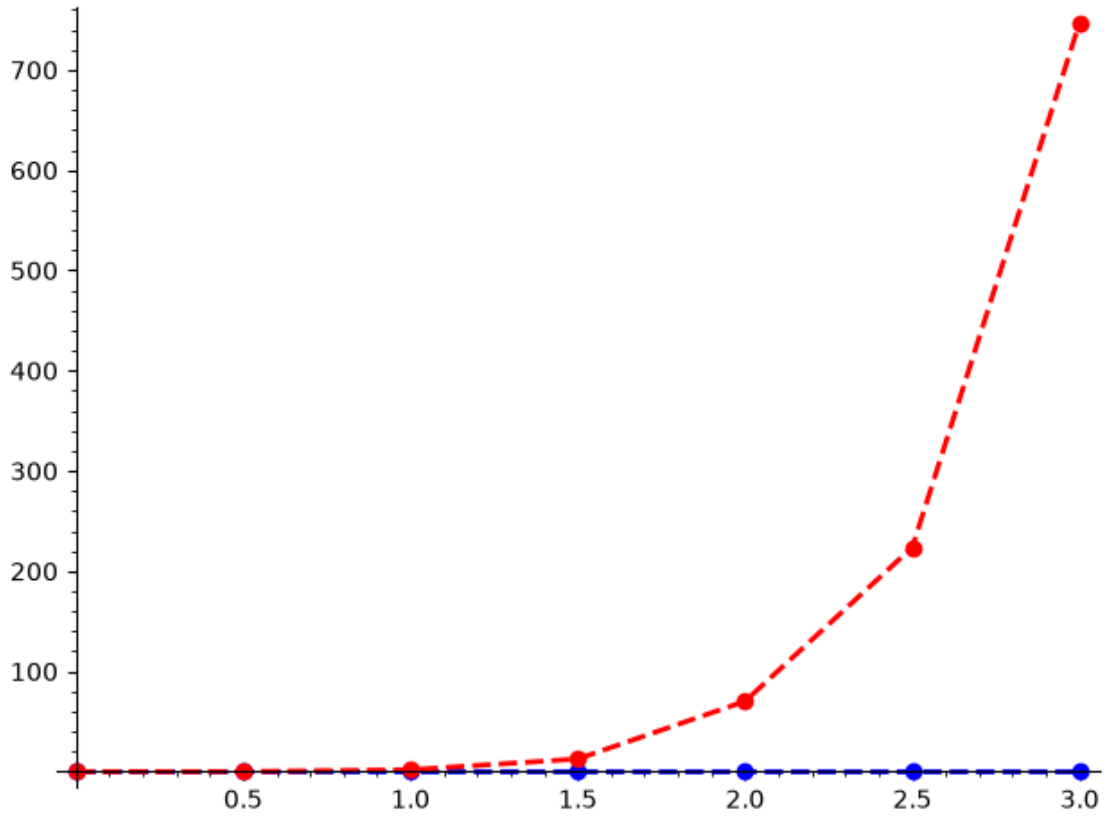
[17]:
```
c_range = [x / 10.0 for x in range(0, 31, 5)] #breaking up the interval [1,3]
K = 100 #how many graphs to sample for each c
n = 500 #size of graphs to sample

def has_spatula(G):
    if spat.is_subgraph(G,induced=False):
        return 1
    else:
        return 0
```

```
def count_spatula(G):
    return G.subgraph_search_count(spat)

has_spat = empirical_means(n,c_range,has_spatula,K,verbose=True,scale=lambda␣
 ↪n,c: c*n^(-5/6))
count_spat =␣
 ↪empirical_means(n,c_range,count_spatula,K,verbose=True,scale=lambda n,c:␣
 ↪c*n^(-5/6))
```

```
Currently sampling with c = 0.000000000000000
Currently sampling with c = 0.500000000000000
Currently sampling with c = 1.00000000000000
Currently sampling with c = 1.50000000000000
Currently sampling with c = 2.00000000000000
Currently sampling with c = 2.50000000000000
Currently sampling with c = 3.00000000000000
Currently sampling with c = 0.000000000000000
Currently sampling with c = 0.500000000000000
Currently sampling with c = 1.00000000000000
Currently sampling with c = 1.50000000000000
Currently sampling with c = 2.00000000000000
Currently sampling with c = 2.50000000000000
Currently sampling with c = 3.00000000000000
```

[20]:
```
line(has_spat,thickness=2,linestyle='--',marker='o') +␣
 ↪line(count_spat,thickness=2,linestyle='--',marker='o',color="red")
```

[20]:


700
600
500
400
300
200
100
0.5
1.0
1.5
2.0
2.5
3.0