16. Statistics Basic

In [1]:
```
%matplotlib inline
import networkx as nx
import numpy as np
import matplotlib.pylab as plt
import matplotlib.axes  as axes
import math
import scipy.optimize as opt
import scipy.stats as stats
import netlab as nl
```

Confidence Interval

In [2]:
```
a = np.random.normal(loc=120, scale=1.5, size=25)
print (a)
```

```
[120.52360591 120.65548198 119.22991278 118.09107671 119.13649566
 121.79530688 118.7785582  121.49600567 120.56963384 118.92190098
 118.7548623  118.93454761 118.30938898 121.03437658 119.7735088
 119.33476249 123.31843362 116.57135298 120.27086975 119.96745255
 119.69261621 120.64636377 120.39631637 119.28467209 119.16285545]
```

In [3]:
```
n = len(a)
mean = np.mean(a)
sigma = np.std(a)
print ('(%.3f, %.3f, %d)' % (mean, sigma, n))
```

```
(119.786, 1.339, 25)
```

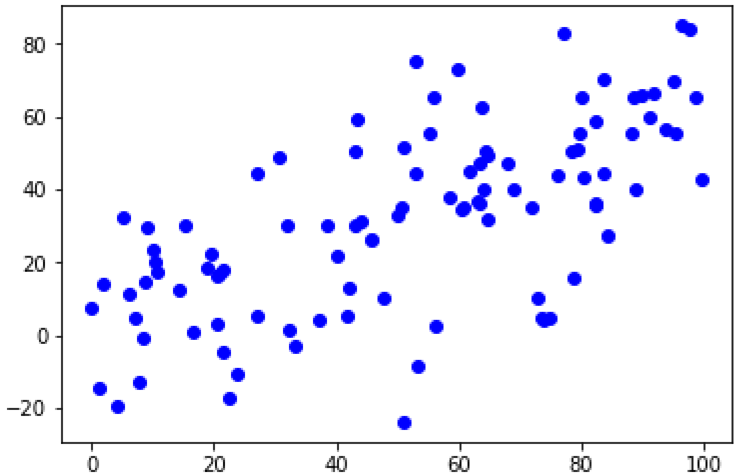
In [4]:
```
confidence_level = 95
confidence_coeff = confidence_level / 100.0
confidence_interval = stats.norm.interval(confidence_coeff, loc=mean, scale=
print ('%.3f %.3f' % (confidence_interval[0], confidence_interval[1]))
```

```
119.261 120.311
```

Simple Linear Regresssion

In [5]:
```
data = np.loadtxt('Data\y11.txt')
x = data[:,0]
y = data[:,1]
plt.plot(x, y, 'bo')
```

Out[5]: [<matplotlib.lines.Line2D at 0x250c53ad080>]



Linear function

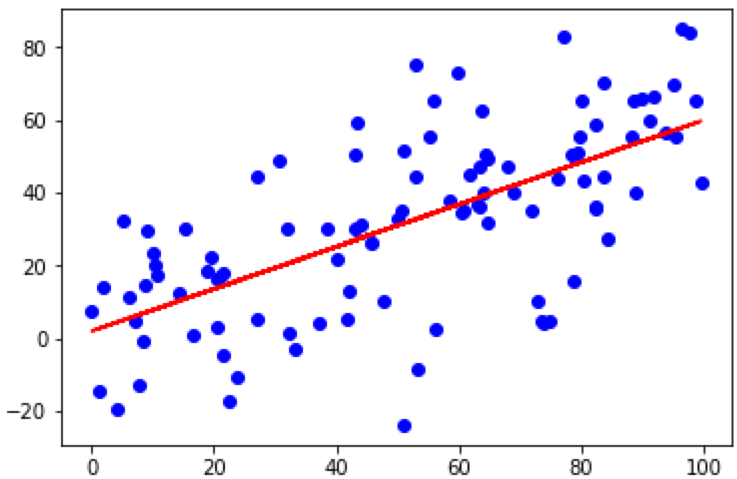
$$f(x) = ax + b$$

In [6]:

```
a, b, corr, p, stderr = stats.linregress(x,y)
z = a * x + b
print ('f(x) = %f x %c %f' % (a, '+' if b >= 0 else '-', abs(b)))
print ('corr    = %f' % corr)
print ('p       = %f' % p)
print ('std err = %f' % stderr)
plt.plot(x, y, 'bo')
plt.plot(x, z, 'r')
```

```
f(x) = 0.579406 x + 1.958615
corr    = 0.660108
p       = 0.000000
std err = 0.066603
```

Out[6]: [<matplotlib.lines.Line2D at 0x250c540b5f8>]



In [7]:

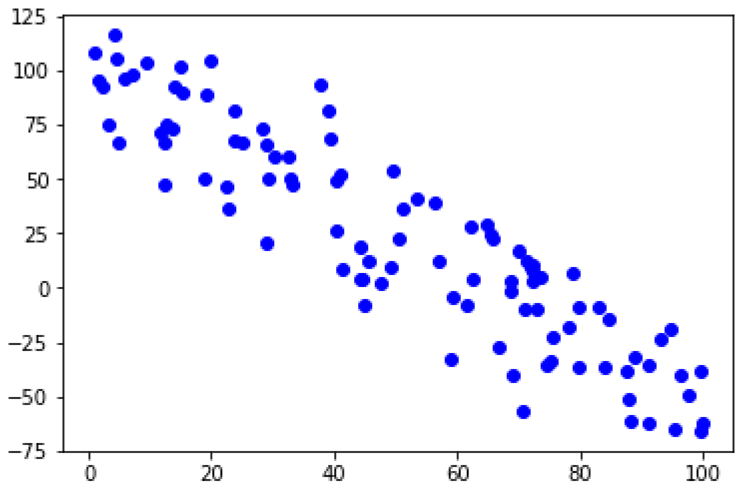
```

stats.linregress??
```

In [8]:

```
data = np.loadtxt('Data\y12.txt')
x = data[:,0]
y = data[:,1]
plt.plot(x, y, 'bo')
```

Out[8]:

```
[<matplotlib.lines.Line2D at 0x250c6ac04e0>]
```



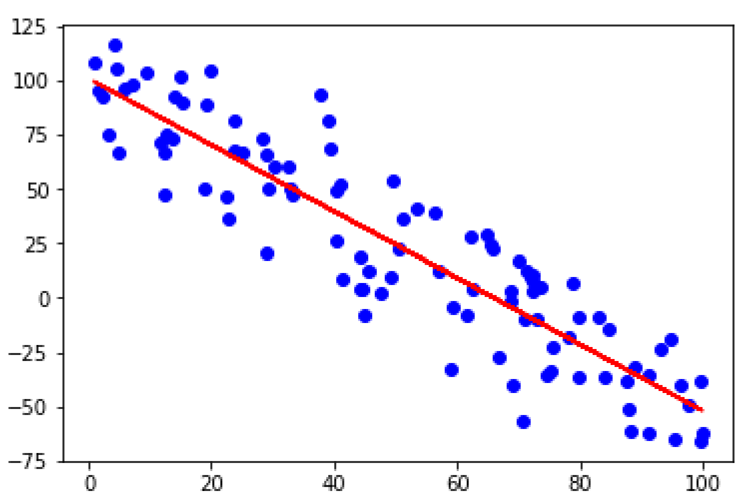
In [9]:

```
a, b, corr, p, stderr = stats.linregress(x,y)
z = a * x + b
print ('f(x) = %f x %c %f' % (a, '+' if b >= 0 else '-', abs(b)))
print ('corr    = %f' % corr)
print ('p       = %f' % p)
print ('std err = %f' % stderr)
plt.plot(x, y, 'bo')
plt.plot(x, z, 'r')
```

```
f(x) = -1.528676 x + 100.858129
corr    = -0.910921
p       = 0.000000
std err = 0.069941
```

Out[9]:

```
[<matplotlib.lines.Line2D at 0x250c6aef908>]
```



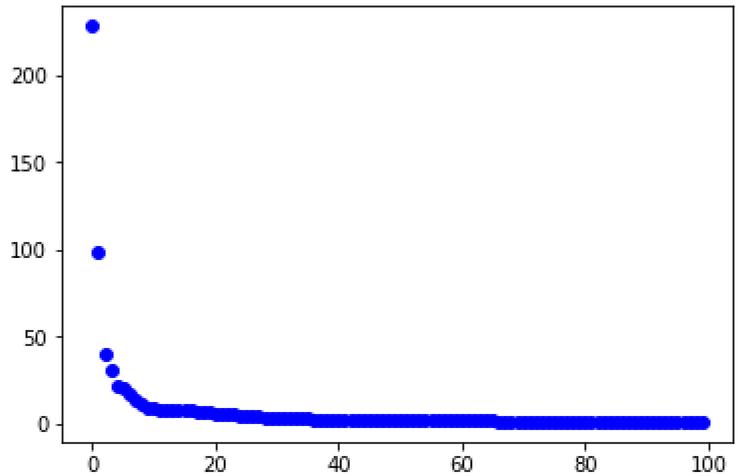
Nonlinear Regression Using Linear Base Functions

In [10]:

```python
data = np.loadtxt('Data\yp.txt')
x = data[:,0]
y = data[:,1]
plt.plot(x, y, 'bo')
```

Out[10]:

```
[<matplotlib.lines.Line2D at 0x250c6b71d30>]
```



Log-log linear function

consider a function as

$$f(x) = bx^a$$

Take logarithm on both sides:

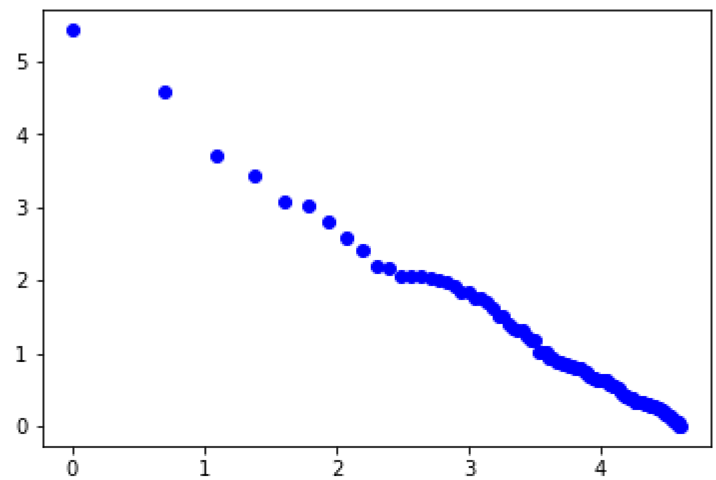
$$\begin{aligned} \log f(x) &= \log b + \log x^a \\ &= \log b + a \log x \end{aligned}$$

By properly substituting:

$$F(x) = B + AX$$

In [11]:
```
x = range(1,101)
xlog = np.log(x)
ylog = np.log(y)
plt.plot(xlog, ylog, 'bo')
```

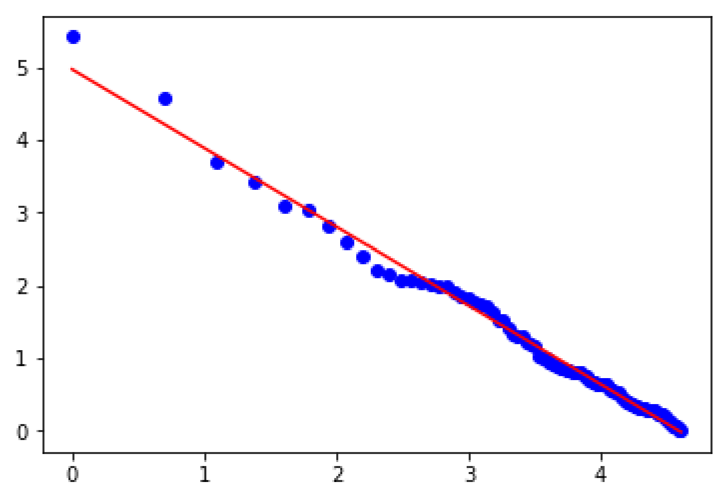
Out[11]:
```
[<matplotlib.lines.Line2D at 0x250c6bca438>]
```



In [12]:
```
a, b, corr, p, stderr = stats.linregress(xlog,ylog)
zlog = a * xlog + b
print ('corr=', corr)
print ('err =', stderr)
plt.plot(xlog, ylog, 'bo')
plt.plot(xlog, zlog, 'r')
```

```
corr= -0.995468410955407
err = 0.010470269575618493
```

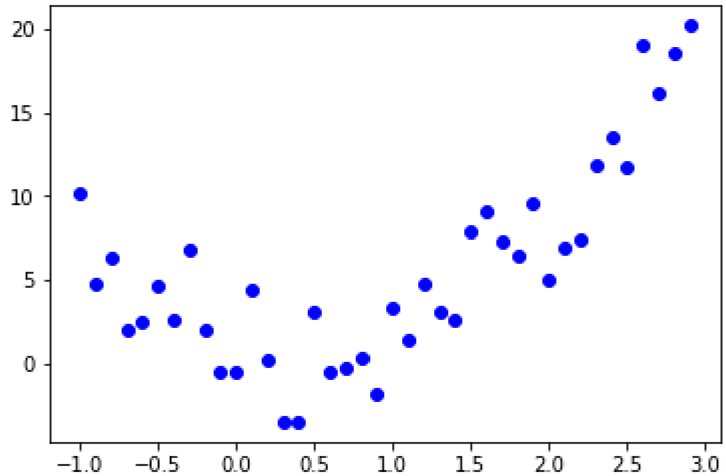
Out[12]:
```
[<matplotlib.lines.Line2D at 0x250c6be8cf8>]
```



Nonlinear regression with Optimization

In [13]:
```
data = np.loadtxt('Data\y2.txt')
x = data[:,0]
y = data[:,1]
plt.plot(x, y, 'bo')
```

Out[13]: [<matplotlib.lines.Line2D at 0x250c6c76630>]



Quadratic function

$$f(x) = ax^2 + bx + c$$

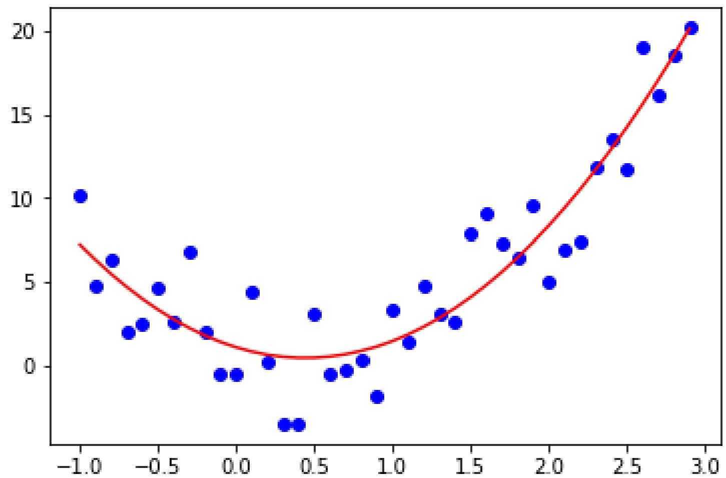
In [14]:
```
def quadratic(x,a,b,c):
    return  a * x ** 2 + b * x + c
```

In [15]:
```
popt,pcov = opt.curve_fit(quadratic, x, y)
print ('pcov=\n', pcov)
print ('stdev err=', np.sqrt(np.diag(pcov)))
z = quadratic(x, *popt)
```

```
pcov=
 [[ 0.10035442 -0.1906734  -0.0431524 ]
 [-0.1906734   0.46905656 -0.01944869]
 [-0.0431524  -0.01944869  0.25720235]]
stdev err= [0.31678766 0.68487704 0.50715121]
```

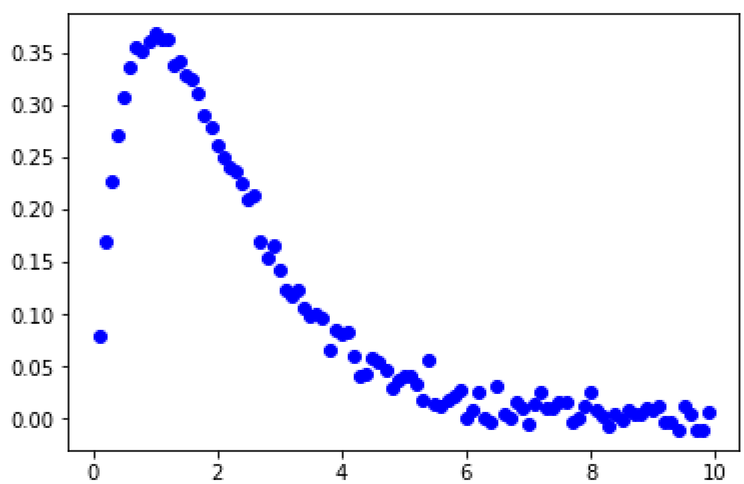
In [16]:
```
plt.plot(x, y, 'bo')
plt.plot(x, z, 'r')
```

Out[16]: [<matplotlib.lines.Line2D at 0x250c6c9ed30>]



In [17]:
```
data = np.loadtxt('Data\y7.txt')
x = data[:,0]
y = data[:,1]
plt.plot(x, y, 'bo')
```

Out[17]: [<matplotlib.lines.Line2D at 0x250c6d38a90>]



Gammar function

$$f(x) = ax^b e^{cx}$$

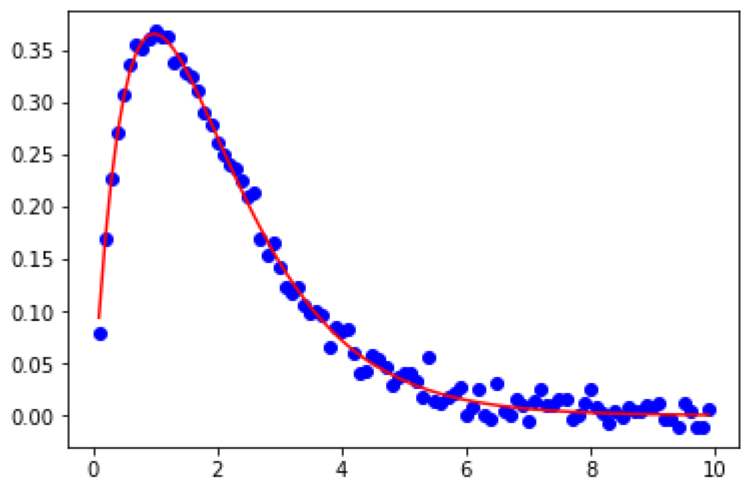
In [18]:
```
def gamma(x, a, b, c):
    return  ( a * x ** b ) * np.exp( c * x )
```

In [19]:
```
popt,pcov = opt.curve_fit(gamma, x, y)
print ('popt=\n', popt)
print ('pcov=\n', pcov)
print ('stdev err=', np.sqrt(np.diag(pcov)))
z = gamma(x, *popt)
```

```
popt=
 [ 0.98708091  0.97783386 -0.99144034]
pcov=
 [[ 0.00050611  0.00046711 -0.0003879 ]
 [ 0.00046711  0.00060765 -0.00040746]
 [-0.0003879  -0.00040746  0.00032398]]
stdev err= [0.022497   0.02465048 0.01799935]
```

In [20]:
```
plt.plot(x, y, 'bo')
plt.plot(x, z, 'r')
```

Out[20]:
```
[<matplotlib.lines.Line2D at 0x250c6d66b38>]
```



Degree Fit

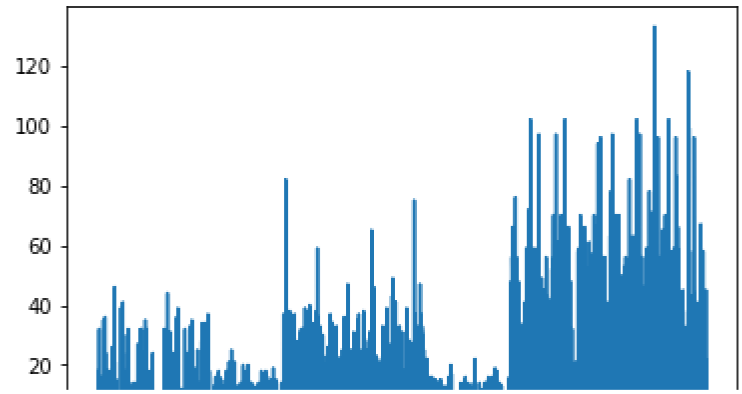
In [21]:

```
# to show the raw data
y = np.loadtxt('Data/degrees.txt', dtype=int)
print ('y =', y)
print ('size: ', y.size)
print ('type: ', y.dtype)
print ('shape:', y.shape)
print ('ndim: ', y.ndim)
plt.plot(y)
```

```
y = [ 1  3  1 ... 45  3 14]
size:  6633
type:  int32
shape: (6633,)
ndim:  1
```

Out[21]:

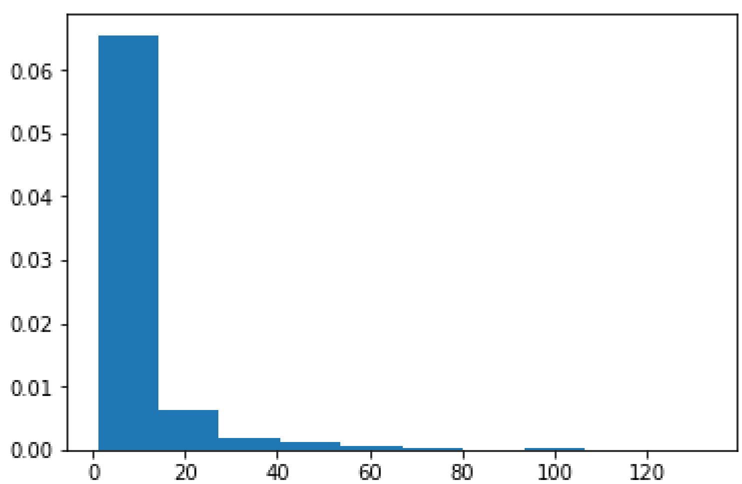
```
[<matplotlib.lines.Line2D at 0x250c6df7fd0>]
```



In [22]:

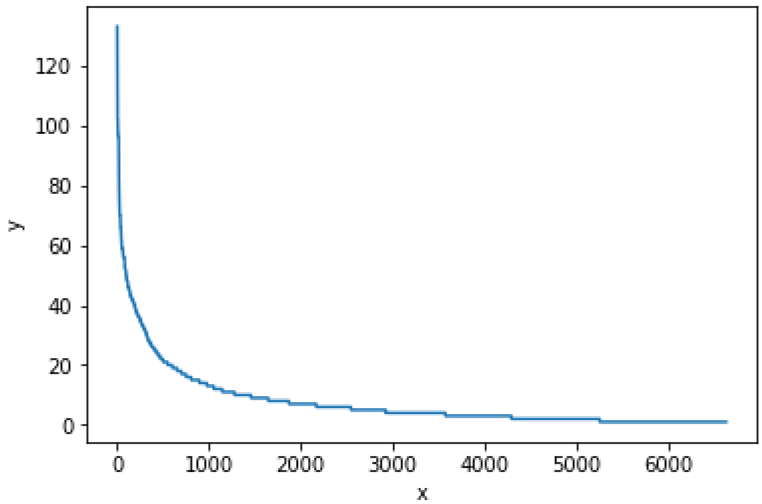
```
# to show the histogram
plt.hist(y, normed=True)
print ('# of points =', len(y))
```

```
# of points = 6633

C:\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning: The
'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



In [23]:
```
y = sorted(y, reverse=True)
x = np.arange(1,len(y)+1)
# to show the rank
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('y')
```

Out[23]:
```
Text(0,0.5,'y')
```



In [24]:
```
def power_law(x,a,c):
    return c * (x ** -a)
```

In [25]:
```
# fail because too big x values
popt,pcov = opt.curve_fit(power_law, x, y)
```

```
C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: RuntimeWarning: overflo
w encountered in power

C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: RuntimeWarning: overflo
w encountered in multiply
```

In [26]:
```
def compare(x,y,z):
    plt.loglog(x,y, marker='o')
    plt.plot(x,z)
    plt.show()
```

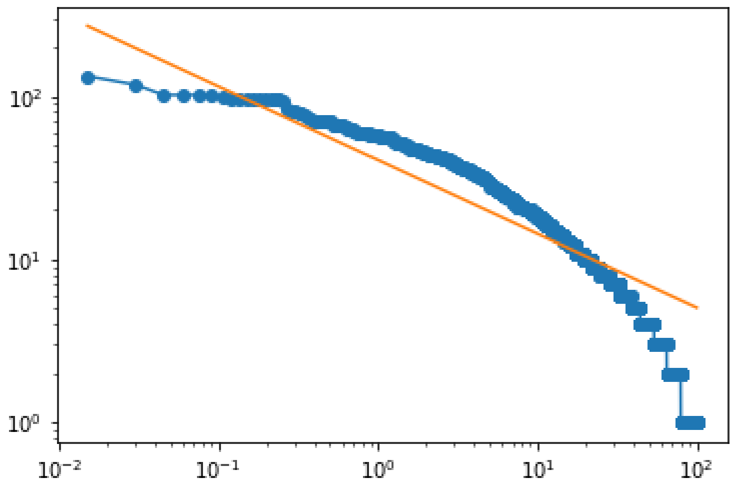
Scaling x-axis

The x-axis, or the rank, needs not be an enumerable counter. Instead, you may think the x-axis as a percentile so that it is scaled from 0 to 100.

In [27]:

```
delta = 1.0 / float(len(x)) * 100
x = np.arange(delta, 100.0+delta, delta)

print ('\nfitting to power law')
# fail in curve_fit due to a bad initial value
popt,pcov = opt.curve_fit(power_law, x, y)
print ('pcov=\n', pcov)
print ('stdev err=', np.sqrt(np.diag(pcov)))
z = power_law(x, *popt)
compare(x,y,z)
```

```
fitting to power law
pcov=
 [[3.45916823e-06 8.56050347e-05]
 [8.56050347e-05 4.20577177e-02]]
stdev err= [0.00185988 0.20507978]
```



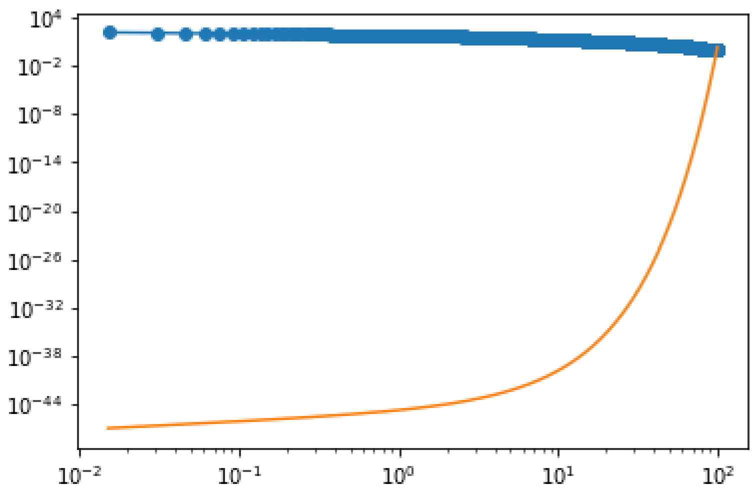
In [28]:

```
def power_exp(x,a,b,c):
    return c * (x ** a) * np.exp(b * x)
```

In [29]:

```
print ('\nfitting to power law with arbitrary')
popt,pcov = opt.curve_fit(power_exp, x, y)
print ('popt=\n', popt)
print ('pcov=\n', pcov)
print ('stdev err=', np.sqrt(np.diag(pcov)))
z = power_exp(x, *popt)
compare(x,y,z)
```

```
fitting to power law with arbitrary
popt=
 [9.99966593e-01 9.99993868e-01 7.38921955e-46]
pcov=
 [[ 2.19813968e+09 -2.22005881e+07 -5.84523088e-36]
 [-2.22005881e+07  2.24225369e+05  5.90347553e-38]
 [-5.84523088e-36  5.90347553e-38  1.55435036e-80]]
stdev err= [4.68843223e+04 4.73524412e+02 1.24673588e-40]
```



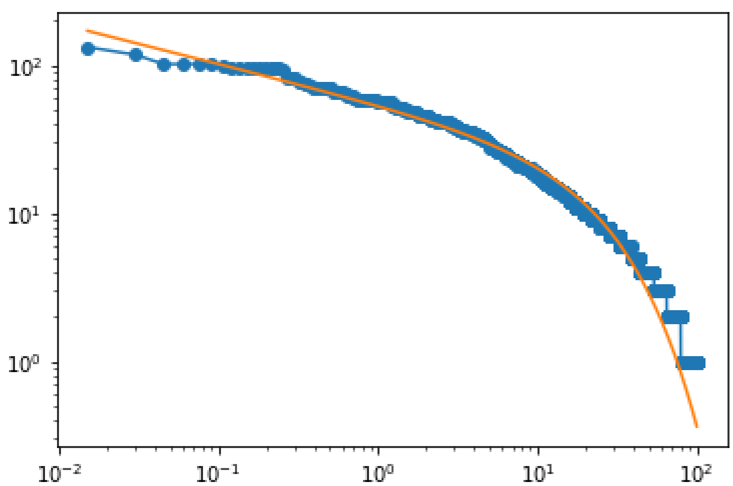
Intial Values

Decide intial values of parameters when they are clear and present. In the above example, we know it is a decreasing function. So, a and b should be nagative.

In [30]:

```
print ('\nfitting to power law with exponential cutoff with an initial value
popt,pcov = opt.curve_fit(power_exp, x, y, p0=[-2, -2, 10])
print ('popt=\n', popt)
print ('pcov=\n', pcov)
print ('stdev err=', np.sqrt(np.diag(pcov)))
z = power_exp(x, *popt)
compare(x,y,z)
```

```
fitting to power law with exponential cutoff with an initial value
popt=
 [-2.71764262e-01 -3.76132009e-02  5.47880810e+01]
pcov=
 [[ 9.24605466e-07 -1.35387381e-07  1.18384187e-05]
 [-1.35387381e-07  3.99241869e-08 -8.67196868e-06]
 [ 1.18384187e-05 -8.67196868e-06  7.45344200e-03]]
stdev err= [0.00096156 0.00019981 0.08633332]
```



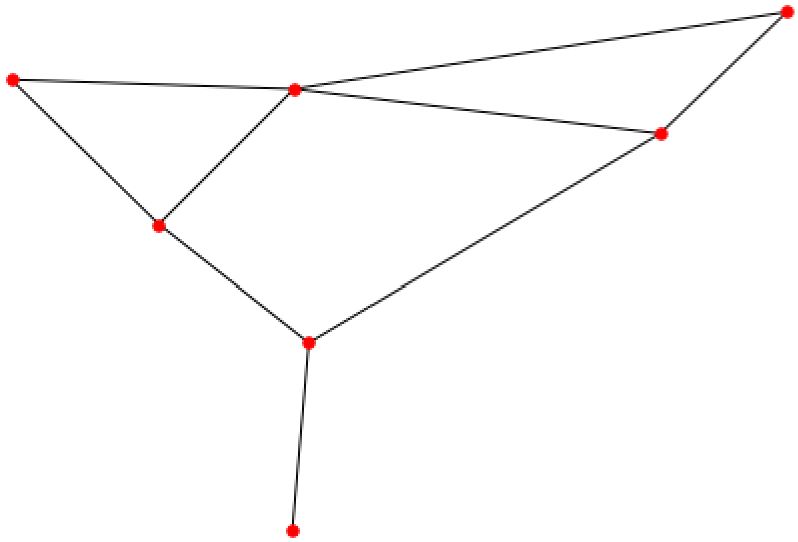
Visualization of Degree Rank Analysis and Fitting

In [41]:

```
symbol = nl.marker()

def degree_analysis(G, name=None):
    y = sorted(list(dict(nx.degree(G)).values()), reverse=True)
    x = range(1, len(y)+1, 1)
    m0, m1 = next(symbol)
    popt, pcov = opt.curve_fit(power_law, x, y)
    print ('nonlinear_regression\npopt=%s\npcov=\n%s\n' % (popt, pcov))
    z = power_law(x, *popt)

    plt.loglog(x, y, m0, label=name+' data')
    plt.loglog(x, z, m1, label=name+' fitted')

```

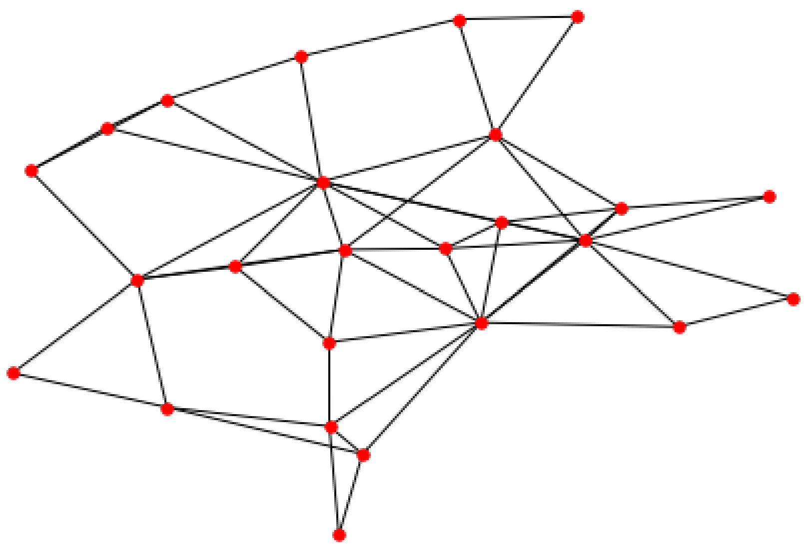
In [42]:

```
f1 = 'Graphs/g35.gml'
G1 = nx.read_gml(f1)
nx.draw(G1, node_size=30)
```

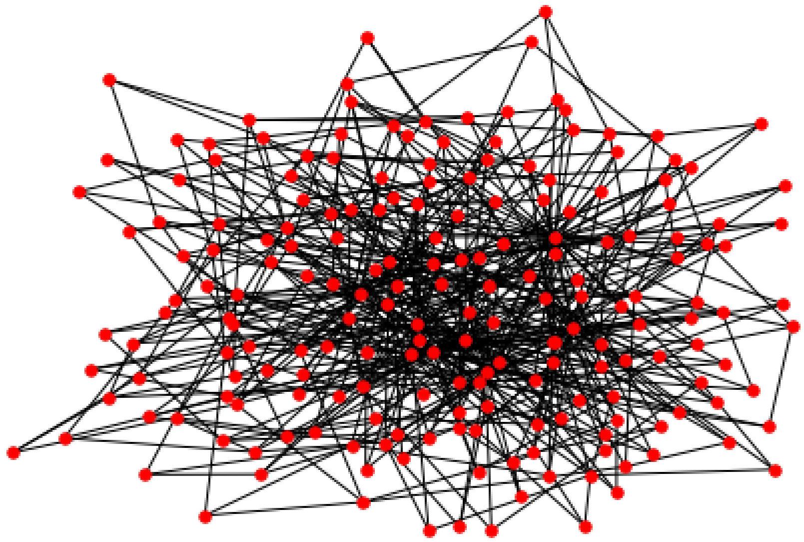


In [43]:

```
f2 = 'data/AttMpls.gml'
G2 = nx.read_gml(f2)
nx.draw(G2, node_size=30)
```

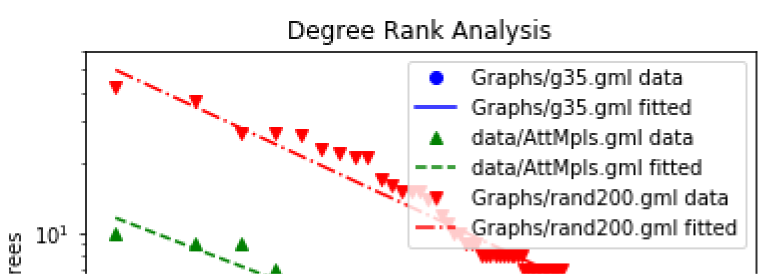


In [44]:
```
f3 = 'Graphs/rand200.gml'
G3 = nx.read_gml(f3)
nx.draw(G3, node_size=30)
```



In [45]:
```
degree_analysis(G1, f1)
degree_analysis(G2, f2)
degree_analysis(G3, f3)

plt.xlabel('rank')
plt.ylabel('degrees')
plt.title('Degree Rank Analysis')
plt.legend()
plt.show()
```

```
nonlinear_regression
popt=[ 0.52745048 49.9506148 ]
pcov=
[[3.57145516e-05 3.96954104e-03]
 [3.96954104e-03 6.87510514e-01]]
```



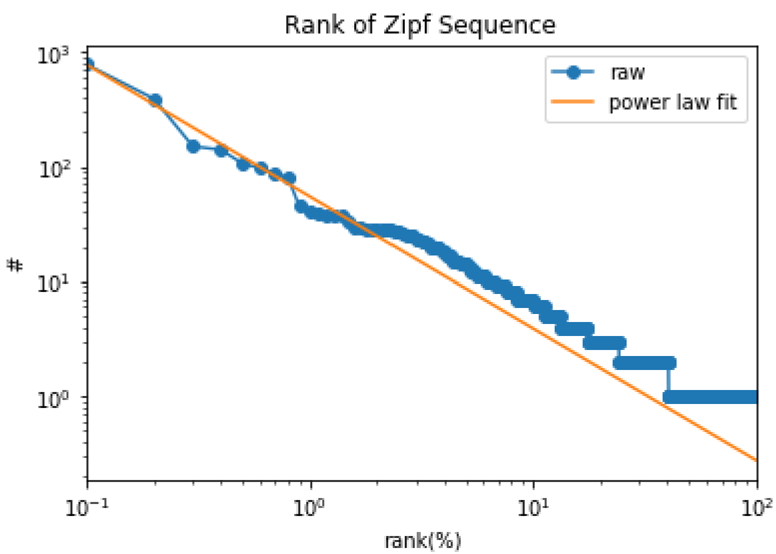
Advanced Fitting

In [47]:
```
def power_cumulative(x,a,c):
    if a != 1.0:
        return c/(1-a) * x ** (-a+1)
    else:
        return c * np.log(x)

def draw_xy(x,y):
    plt.title('Rank of Zipf Sequence')
    plt.ylabel('#')
    plt.xlabel('rank(%)')
    plt.xlim([min(x), max(x)])
    plt.loglog(x,y,marker='o', label='raw')
    return

def fitting_power(x,y):
    popt, pcov = opt.curve_fit(power_law, x, y)
    print ('coeff =', popt)
    z = power_law(x, *popt)
    plt.plot(x, z, label='power law fit')
    return

def fitting_log_bin(x,y):
    x1 = []
    y1 = []
    si = 1.0
    while si <= len(x):
        i = int(si)
        x1.append(x[i])
        y1.append(y[i])
        si *= 1.2
    popt, pcov = opt.curve_fit(power_law, x1, y1)
    print ('log bin coeff =', popt)
    z1 = power_law(x1, *popt)
    plt.plot(x1, z1, marker='s', label='log bin fit')
    return

def fitting_cumulative(x,y):
    y1 = y
    for i in range(1,len(y1)):
        y1[i] += y1[i-1]

    popt, pcov = opt.curve_fit(power_cumulative, x, y1, p0=[0.5, 1.0])
    z1 = power_cumulative(x, *popt)
    z = np.zeros([len(z1)])
    z[0] = z1[0]
    for i in range(1,len(z1)):
        z[i] = z1[i] - z1[i-1]
    plt.plot(x,z,label='cum. fit')
    return
```

In [55]:
```
s = [nx.utils.random_sequence.zipf_rv(alpha=2) for i in range(1000)]
y = sorted(s, reverse=True)
delta = 1.0 / float(len(y)) * 100
x = np.arange(delta,100.0+delta, delta)
```

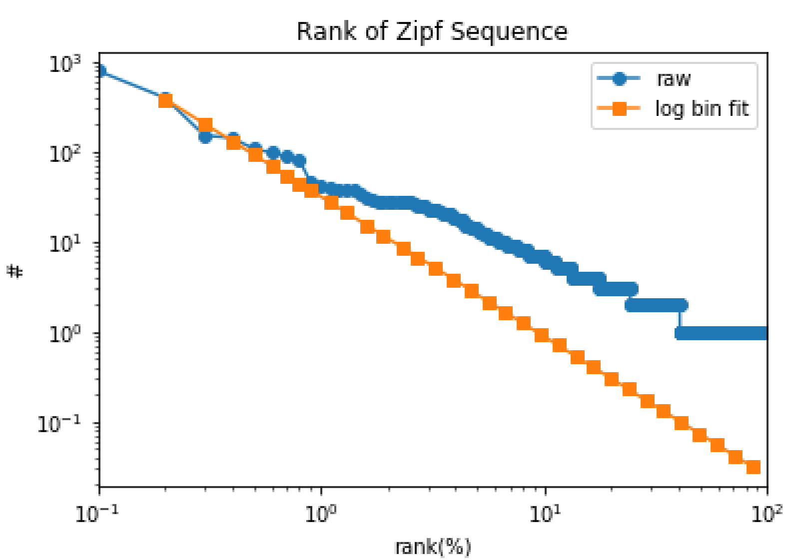
In [58]:
```python
l0 = draw_xy(x,y)
l1 = fitting_power(x,y)
plt.legend()
plt.show()
```

```
coeff = [ 1.15184945 55.18534468]
```

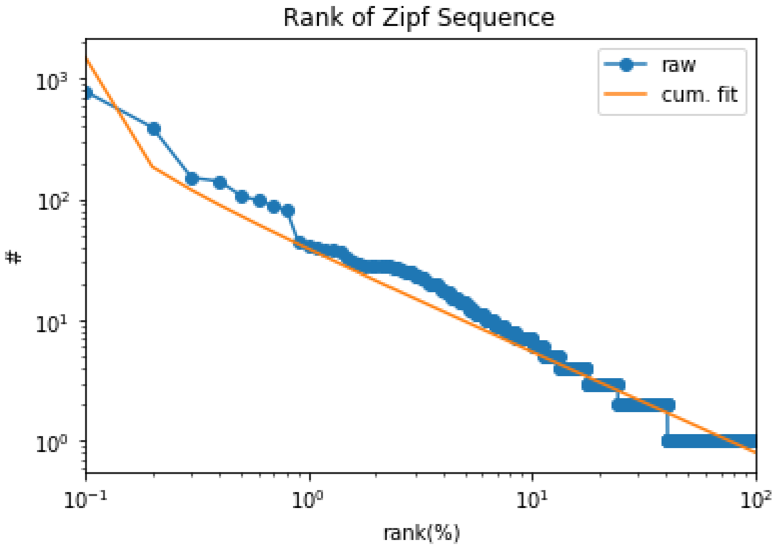


In [59]:
```python
l0 = draw_xy(x,y)
l2 = fitting_log_bin(x,y)
plt.legend()
plt.show()
```

```
log bin coeff = [ 1.54956154 31.44460708]
```

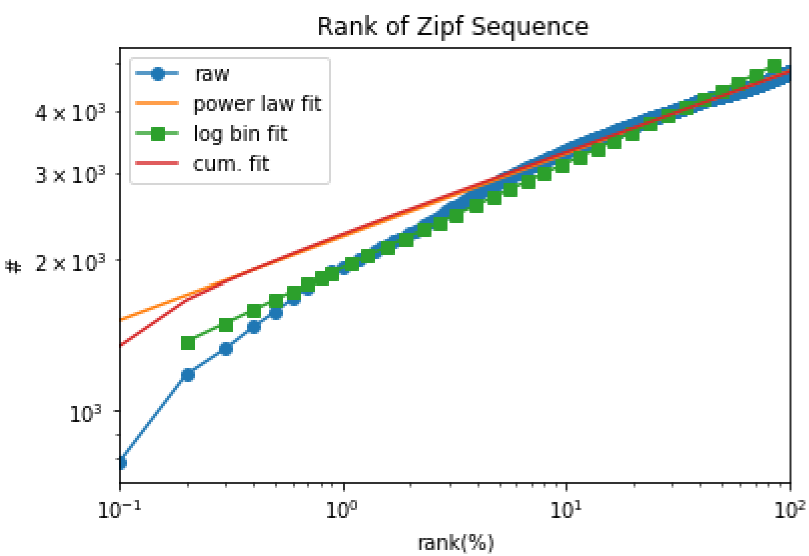


In [60]:
```
l0 = draw_xy(x,y)
l3 = fitting_cumulative(x,y)
plt.legend()
plt.show()
```



In [61]:
```
l0 = draw_xy(x,y)
l1 = fitting_power(x,y)
l2 = fitting_log_bin(x,y)
l3 = fitting_cumulative(x,y)
plt.legend()
plt.show()
```

```
coeff = [-1.66736396e-01  2.22806881e+03]
log bin coeff = [-2.10421651e-01  1.93085887e+03]
```



In [62]:
```
def rank_vertex_degree(G, mini=False):
    D = sorted(nx.degree(G).values(),reverse=True) # degree sequence
    plt.loglog(D,'b-',marker='o')
    plt.title("Degree Rank")
    plt.ylabel("degree")
    plt.xlabel("rank")
    if mini:
        mini_draw(G)
    plt.show()
```

In [ ]:
```

```