Reinforcement Learning: Homework #4 99

Due on April 30, 2020 at 11:59pm

Professor Ziyu Shao

Yiwei Yang
2018533218

Homework4 for Reinforce-Learning SI-252

Yiwei Yang 2018533218

i. Environment Setup

In [380]:
```
import numpy as np
import random
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import multivariate_normal
import random
import math
import matplotlib.pyplot as plt
import sys
from scipy.stats import poisson,beta,expon,kstest,binom
```

ii. Binary Sequences with No Adjacent 1s 10

Problem Background

Consider sequences of length m consisting of 0s and 1s. A sequence is "good" if it has no adjacent 1s. The expected number of 1s in a good sequence if all good sequences are equally likely.

Denotions

For general m, the expected number of 1s is $\mu = \sum_k k\pi_k$, where π_k is the probability that a good sequence of length m has exactly k 1s.

There is no simple closed form expression for μ. So, we use a simulation.

If x is a sequence of 0s and 1s of length m, let $r(x)$ be the number of 1s in the sequence. Assume we can generate a uniformly random sequence of length m with no adjacent 1s. Doing this repeatedly and independently, generating good sequences $Y_1, Y_2, \ldots, Y_n$, we can generate a Monte Carlo estimate of the desired expectation with $\mu \doteq \frac{r(Y_1)+r(Y_2)+\cdots+r(Y_n)}{n}$, $for\ large\ n$. Thus, the problem of estimating μ reduces itself to the problem of simulating a good sequence.

The good sequence generation

The idea is to construct an ergodic Markov chain $X_0, X_1, \ldots$ whose state space is the set of good sequences and whose limiting distribution is uniform on the set of good sequences.

The Markov chain is then generated and, as in the i.i.d. case, we take $\mu \doteq \frac{r(X_1)+r(X_2)+\cdots+r(X_n)}{n}$, $for\ large\ n$, as an MCMC estimate for the desired expectation.

The Markov chain

The Markov chain is constructed as a random walk on a graph whose vertices are good sequences. Proceed as follows.

From a given good sequence, pick one of its m components uniformly at random.

If the component is 1, then switch it to 0. The new sequence is also a good sequence. Move to the new sequence.

If the component is 0, then switch it to 1 only if the resulting sequence is good. If it is, then move to the new sequence. Otherwise, stay put and the walk stays at the current state.

In [3]:
```
m = 100
n = 100000
seq = [0 for x in range(m)]
chainSeq = [seq]
mu = 0
for i in range(n):
    randInt = np.random.randint(0,m)
    if seq[randInt] == 1:
        seq[randInt] = 0
    else:
        if randInt == 0:
            if seq[randInt+1] == 0:
                seq[randInt] = 1
        elif randInt == (m-1):
            if seq[randInt-1] == 0:
                seq[randInt] = 1
        else:
            if seq[randInt+1] == 0 and seq[randInt-1] == 0:
                seq[randInt] = 1
    chainSeq.append(seq)
    mu += sum(seq)
mu /= n
print(mu)
```

```
27.83129
```

The above value is the MCMC estimate for the expected number of 1s, μ. An exact analysis for $m = 100$ gives $\mu = 27.83129$.

iii.Power-Law Distribution 10

Solution For a proposal distribution, we use simple symmetric random walk on the positive integers with reflecting boundary at 1. From 1, the walk always moves to 2. Otherwise, the walk moves left or right with probability 1/2. The proposal chain transition matrix is

$$T_{ij} = \begin{cases} 1/2, & \text{if } j = i \pm 1 \text{ for } i > 1 \\ 1, & \text{if } i = 1 \text{ and } j = 2 \\ 0 & \text{otherwise} \end{cases}$$

The acceptance function is

$$a(i, i+1) = \left(\frac{i}{i+1}\right) \text{ , and } \quad a(i+1, i) = \left(\frac{i+1}{i}\right) \text{ , for } i \geq 2$$

with

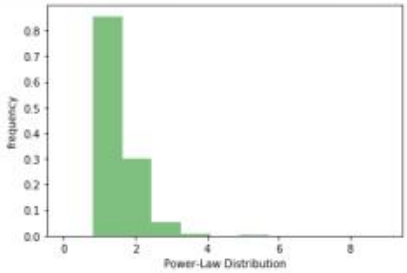
$$a(1,2) = \frac{\pi_2 T_{21}}{\pi_1 T_{12}} = \left(\frac{1}{2}\right)^{3/2} \frac{1}{2} = \left(\frac{1}{2}\right)^{5/2} \quad \text{and} \quad a(2,1) = 2^{5/2}$$

Observe that $a(i+1, i) \geq 1$, for all i The Metropolis-Hastings algorithm can be described as follows. From state $i \geq 2$, flip a fair coin. If heads, go to $i-1$. If tails, set $p = (i/(i+1))^{3/2}$. Flip another coin whose heads probability is p. If heads, go to $i+1$. If tails, stay at i If the chain is at 1, then move to 2 with probability $(1/2)^{5/2} \approx 0.177$. Otherwise, stay at 1.

The chain was run for one million steps.

In [287]:
```
T=1000000
sim=np.zeros(T)
sim[0]=2
for i in range(1,T-1):
    if sim[i-1]==1:
        prob=pow(0.5,2.5)
        new=np.random.choice([1,2], size=1, replace=True, p=[1-prob,prob])
        sim[i]=new
    else:
        prob=pow(0.5,2.5)
        leftright=np.random.choice([-1,1], size=1)
        if leftright==-1:
            sim[i]=sim[i-1]-1
        else:
            sim[i]=np.random.choice([sim[i-1],1+sim[i-1]], size=1,p=[1-prob,prob])
data=sim[1000:T]
```

In [295]:
```
plt.hist(data,11, density=1, facecolor='green', alpha=0.5)
plt.ylabel("frequency")
plt.xlabel("data")
plt.xlabel("Power-Law Distribution")
plt.show()
```



iv. Knapsack Problem 10

- m treasures with labels from 1 to m, where the ,th treasure is worth g_j gold pieces and weighs w_j pounds.
- The maximum weight we can carry is w pounds.
- We must choose a vector $x = (x_1, \ldots, x_m)$, where x_j is 1 if we choose the j th treasure and 0 otherwise, such that the total weight of the treasures j with $x_j = 1$ is at most w
- Let C be the space of all such vectors, so C consists of all binar vectors $(x_1, \ldots, x_m)$ with $\sum_{j=1}^{m} x_j w_j \leq w$
- We wishes to maximize the total worth of the treasure we takes.

The dynamic solution

```
def dynamic(maxWeight, w, v, n):
  start = time.time()

  K = [[0 for x in range(maxWeight+1)] for x in range(n+1)]

  # Build table K[][] in bottom up manner
  for i in range(n + 1):                  # i first items allowed
      for j in range(maxWeight + 1):      # j max wight allowed
          if i == 0 or j == 0:
              K[i][j] = 0
          elif w[i-1] <= j:
              K[i][j] = max(v[i-1] + K[i-1][j-w[i-1]],  K[i-1][j])
          else:
              K[i][j] = K[i-1][j]

  best = K[n][maxWeight]
  elapsed = time.time() - start
  return best, elapsed
```

The MCMC solution

We can go from any $x \in C$ to $(0, 0, \ldots, 0)$ by dropping treasures one at a time. We can go from $(0, 0, \ldots, 0)$ to any $y \in C$ by picking up treasures one at a time. Combining these, we can go from anywhere to anywhere, so the chain is irreducible. To study periodicity, let's look at some simple cases. First consider the simple case where $w_1 + \cdots + w_m < w$, i.e., the man can carry all the treasure at the same time. Then all binary vectors of length m are allowed. So the period of $(0, 0, \ldots, 0)$ is 2 since, starting at that state, Bilbo needs to pick up and then put down a treasure in order to get back to that state. In fact, if Bilbo starts at $(0, 0, \ldots, 0)$, after any odd number of moves he will be carrying an odd number of treasures.

Now consider the case where $w_1 > w$, i.e., the first treasure is too heavy. From any $x \in C$, there is a $1/m$ chance that the chain will try to pick up the first treasure, and if that happens, the chain will stay at x. So the period of each state is 1.

Algorithm

We can apply Metropolis-Hastings using the chain from (a) to make proposals. Start at $(0, 0, \ldots, 0)$. Suppose the current state is $x = (x_1, \ldots, x_m)$. Then:

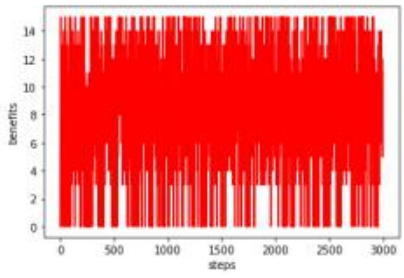
1. Choose a uniformly random J in $\{1, 2, \ldots, m\}$, and obtain y from x by replacing x_J with $1 - x_J$
2. If y is not in C, stay at x. If y is in C, flip a coin that lands Heads with probability $\min\left(1, e^{\beta(V(y)-V(x))}\right)$. If the coin lands Heads, go to y otherwise, stay at x

This chain will converge to the desired stationary distribution. But how should β be chosen? If β is very large, then the best solutions are given very high probability, but the chain may be very slow to converge to the stationary distribution since it can easily get stuck in local modes: the chain may find itself in a state which, while not globally optimal, is still better than the other states that can be reached in one step, and then the probability of rejecting proposals to go elsewhere may be very high. On the other hand, if β is close to 0, then it's easy for the chain to explore the space, but there isn't as much incentive for the chain to uncover good solutions.

In [467]:
```
def knapsack(times):
    #initialization
    random.seed(123)
    weight=[2,2,6,5,4]
    value=[6,3,5,4,6]
    V=10
    i=1
    l=1
    selection=np.zeros(5)
    candidates=np.zeros((times,5))
    benefits=np.zeros(times)
    total_weight=np.zeros(times)
    selection_new=selection
    chooseitem=np.arange(5)

    #Selection Loop: here we repeatedly propose and accept/reject scenarios to take on the trip
    for m in range(times):
        #here we tune the algorithm to help produce more exploration in the cases where the chain gets stuck
        if m/1>500:
            l=0.15
        else:
            l=1

        #Generation/Proposal Step: Here we randomly select a bit to flip
        j=np.random.choice(chooseitem, size=1)
        selection_new=selection
        if selection[j]==1:
            selection_new[j]=0
        else:
            selection_new[j]=1

        #Accept/Reject Step: Here we either advance or fail to advance our Markov
        #chain based on the acceptance criteria that the weight be less than V
        #and the choice meets the Metropolis-Hasting criteria.
        if np.sum(weight*selection_new)>V:
            selection=selection
        if np.sum(weight*selection_new)<=V and np.random.binomial(1,min(1,np.exp(l*(np.sum(selection_new*value)-np.sum(selection
            selection=selection_new;
            candidates[i]=selection_new;
            benefits[i]=np.sum(value*selection_new);
            total_weight[i]=np.sum(weight*selection_new);
            i=i+1
    temp=np.arange(times)
    plt.plot(temp[:3000],benefits[:3000],'r-')
    plt.ylabel("benefits")
    plt.xlabel("steps")
    print("The max value is ",np.max(benefits),"\n")
    print("The selection string is: ",candidates[np.where(benefits==np.max(benefits))[0][0]],"\n")
```

In [468]:
```
knapsack(10000)
```

```
The max value is  15.0

The selection string is:  [1. 1. 0. 0. 1.]
```



Comments on the MCMC of Knapsack

As we can see om the graph, it goes to astringent quickly after the starting point. and roughly takes 3000 times to get the results. However, the selection is not always the right one for it's the heuristic algorithm. Overall, it's far more better than the dynamic solution. For optimization, the sub-optimized situation can be sheered during the main loop. this can speed up about 30% of the performance.

v. Standard Normal Distribution

9

Solution Write the standard normal density function as $\pi_z = e^{-z^2/2}/\sqrt{2\pi}$. For the Metropolis-Hastings proposal distribution, we choose the uniform distribution on an interval of length two centered at the current state. From state s, the proposal chain moves to t, where t is uniformly distributed on $(s-1, s+1)$. Hence, the conditional density given s is constant, with

$$\Gamma_{st} = \frac{1}{2}, \text{ for } s-1 \le t \le s+1$$

The acceptance function is $\alpha(s,t) = \frac{\pi_t \Gamma_{ts}}{\pi_s \Gamma_{st}} = \left(\frac{e^{-t^2/2}}{\sqrt{2\pi}}\right)(1/2) / \left(\frac{e^{-s^2/2}}{\sqrt{2\pi}}\right)(1/2) = e^{-\left(t^2-s^2\right)/2}$ Notice that the length of the interval for the uniform proposal distribution does not affect the acceptance function.

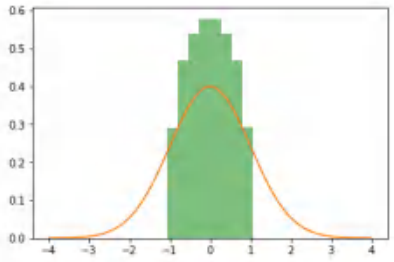
M-H Algorithm

Because the np.random.uniform only accept `int` variables, so the final graph may not be so acurate.

Let $f(x)$ be a function that is proportional to the desired probability distribution $P(x)$ (a.k.a. a target distribution).

- update $\mathcal{X}$ from the π_t
  - sample $p^* \sim f_p\left(p \mid p^{(t)}\right)$
  - applying the acceptance function $a(t, t) = e^{-\left(t^2 - t^2\right)/2}$

In [360]:
```
T=100000
sim=np.zeros(T)
state=0
for i in range(1,T-1):
    prop=np.random.uniform(1,int(state)-1,int(state)+1)
    acc=np.exp(-(pow(prop,2)-pow(state,2))/2)
    if np.random.uniform()<acc:
        state=prop
    sim[i]=state
plt.hist(sim,30,range=[-4,4],density=1, facecolor='green', alpha=0.5)
x = np.linspace(-4, 4, 1000)
pdf = norm().pdf(x)
plt.show()
```



这个仿真结果与pdf有较大差距，请注意检查代码。

Box-Muller Algothm

The Box-Muller transform is a neat little "trick" that allows us to sample from a pair of normally distributed variables using a source of only uniformly distributed variables. The transform is actually pretty simple to compute. Given two independent uniformly distributed random variables U_{1}, U_{2} on the interval $(0,1),$ we define two new random variables, R and $\Theta,$ that intuitively representing polar coordinates as such: $R=\sqrt{-2 \ln U_{1}}$ $\Theta=2 \pi U_{2}$ Now using the standard transformation from polar coordinates R, Θ to Cartesian ones $X, Y,$ we claim that X and Y are independent standard normally distributed random variables: $X=R \cos \Theta \quad=\sqrt{-2 \ln U_{1}} \cos \left(2 \pi U_{2}\right)$ $Y=R \sin \Theta \quad=\sqrt{-2 \ln U_{1}} \sin \left(2 \pi U_{2}\right)$ Let's take a look at the proof to gain some intuition on how this works. Proof Starting with $U_{1}, U_{2},$ let's see what kind of distributions we have for $R, \Theta .$ It should be clear that Θ is also uniformly distributed since it's just multiplying by a constant (2π), but let's go through the motions to explicitly see that. From the CDF of Θ: $$ \begin{aligned} P(\Theta \leq \theta) &=P\left(2 \pi U_{2} \leq \theta\right) \\ &=P\left(U_{2} \leq \frac{\theta}{2 \pi}\right) \\ \Theta & \sim U n i f o r m(0,2 \pi) \end{aligned} $$

So we have our first result that Θ is uniformly distributed on $(0,2 \pi)$ (as we would expect). Using a more explicit method, we can find the distribution of R. We know that the mapping from U_{1} to R is one-to-one, which we'll call g (i.e. $R=g\left(U_{1}\right)$). So if we're trying to find the probability of R over $\left[r_{1}, r_{2}\right]$ (by integrating its PDF), there is some equivalent range over $\left(g^{-1}\left(r_{1}\right), g^{-1}\left(r_{2}\right)\right)$ where we can integrate over the PDF of U_{1}. Let's see how this works : $P\left(r_{1} \leq R \leq r_{2}\right)=\int_{g^{-1}\left(r_{2}\right)}^{g^{-1}\left(r_{1}\right)} f_{U_{1}}(s) d s \quad$ since $g(x)$ is strictly decreasing $$ \begin{array}{ll} =-\int_{g^{-1}\left(r_{1}\right)}^{g^{-1}\left(r_{2}\right)} d s & \text { since } f_{U_{1}}(u)=1 \\ =-\int_{r_{1}}^{r_{2}}-1 e^{\frac{-t^{2}}{2}} d t & \text { change variables } t=\sqrt{-2 \ln s} \\ f_{R}(r)= & r e^{\frac{-r^{2}}{2}} \end{array} $$ giving us the PDF for $R, f_{R}(r)^{[4] \text { . }}$ It should also be clear that R and Θ are independent because U_{1} and U_{2} are independent.

Now that we have the distributions for both R and $\Theta,$ we can apply the same procedure as above using variable substitution (for multiple variables) to derive the joint distribution of X and Y (we'll represent the area in the transformed space by $A^{[5]}$). $$ \begin{aligned} P\left(x_{1}\right.&\left. \leq X \leq x_{2}, y_{1} \leq Y \leq y_{2}\right) \\ &=\iint_{A} f_{R, \Theta}(u, v) d u d v \end{aligned} $$ $=\iint_{A} f_{R}(u) f_{\Theta}(v) d u d v \quad$ since R and Θ are independent $$ =\iint_{A} u e^{\frac{-u^{2}}{2}} \frac{1}{2 \pi} d u d v $$ At this point, we should remember that if $p=u \cos (v), q=u \sin (v),$ then solving for u, v results in $u=\sqrt{q^{2}+p^{2}}, v=\arctan \left(\frac{q}{p}\right) .$ Also, substitution for multiple variables means that $d u d v=\left|\operatorname{det}\left(\frac{\partial(u, v)}{\partial(p, q)}\right)\right| d p d q,$ plugging our expressions for u and v in:

$$\begin{aligned} d u d v &=\left|\operatorname{det}\left(\frac{\partial(u, v)}{\partial(p, q)}\right)\right| d p d q \\ &=\left|\begin{array}{cc}\frac{\partial u}{\partial p} & \frac{\partial u}{\partial q} \\ \frac{\partial v}{\partial p} & \frac{\partial v}{\partial q}\end{array}\right| d p d q \\ &=\left|\begin{array}{cc}\frac{\partial \sqrt{q^{2}+p^{2}}}{\partial p} & \frac{\partial \sqrt{q^{2}+p^{2}}}{\partial q} \\ \frac{\partial \arctan \left(\frac{q}{p}\right)}{\partial p} & \frac{\partial \arctan \left(\frac{q}{p}\right)}{\partial q}\end{array}\right| d p d q \\ &=\left|\begin{array}{c}\frac{p}{\sqrt{p^{2}+q^{2}}} \\ \frac{-q}{p^{2}\left(1+\left(\frac{q}{p}\right)^{2}\right)}\end{array} \frac{q}{p\left(1+\left(\frac{q}{p}\right)^{2}\right)}\end{array}\right| \\ &=\frac{1}{\sqrt{p^{2}+q^{2}}} d p d q \end{aligned}$$

Using the result from $\begin{cases}p=u\cos(v)\\ q=u\sin(v)\end{cases}$, we get:

$P\left(x_{1} \leq X \leq x_{2}, y_{1} \leq Y \leq y_{2}\right)$ $=\int_{x_{1}}^{x_{2}} \int_{y_{1}}^{y_{2}} \frac{\sqrt{p^{2}+q^{2}}}{2 \pi} \cdot e^{\frac{-\left(p^{2}+q^{2}\right)}{2}} \cdot \frac{1}{\sqrt{p^{2}+q^{2}}} d p d q $ $=\int_{x_{1}}^{x_{2}} \underbrace{\frac{1}{\sqrt{2 \pi}} e^{\frac{-p^{2}}{2}}}_{f_{X}(x)} d p \int_{y_{1}}^{y_{2}} \underbrace{\frac{1}{\sqrt{2 \pi}} e^{\frac{-q^{2}}{2}}}_{f_{Y}(y)}$

Equation above shows that X and Y are independent each with PDF matching our standard normal distribution $N(0,1)$ as required.

Implementing Box-Muller Transform

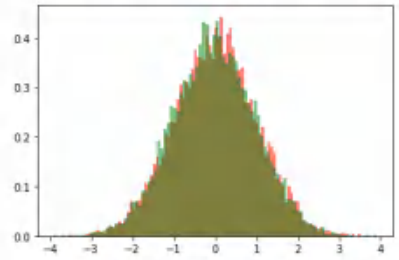
The implementation is a relatively straight forward application :

```
        u1 = random.random()
        u2 = random.random()

    n1 = math.sqrt(-2 * math.log(u1)) * math.cos(2 * math.pi * u2)
    n2 = math.sqrt(-2 * math.log(u1)) * math.sin(2 * math.pi * u2)
    return n1, n2
```

In [407]:
```
N=10000
random.seed(123)
epsilon = sys.float_info.epsilon

# Use KS to test
samples = [box_muller()[0] for x in range(N)]
test_stat, pvalue = kstest(samples, 'norm', args=(0, 1), N=N)
print("sample_N(0,1) vs. N(0, 1): KS=%.4f with p-value = %.4f." % (test_stat, pvalue))

# Plot our samples against our reference distribution
reference = [norm.rvs() for x in range(N)]
plt.hist(samples, 100, density=1, facecolor='red', alpha=0.5)
```

```
plt.hist(reference, 100, density=1, facecolor='green', alpha=0.5)
plt.show()
```

```
sample_N(0,1) vs. N(0, 1): KS=0.0074 with p-value = 0.6372.
```



Comparison between M-H and B-M Algorithm

Compared with an algorithm like B-M Algorithm that directly generates independent samples from a distribution, Metropolis–Hastings and other MCMC algorithms have a number of disadvantages:

- The samples are related. Even if they do follow P (x) correctly in the long run, a set of adjacent samples will be related to each other and cannot reflect the distribution correctly. This means that if we want a group of independent samples, we must discard most of the samples and only take every n samples, for a certain value of n (usually determined by examining the autocorrelation between adjacent samples). Autocorrelation can reduce autocorrelation by increasing the jump width (the average size of the jump width, which is related to the variance of the jump distribution), but it also increases the possibility of rejecting the proposed jump. If the jump size is too large or too small, it will lead to a slow hybrid Markov chain, that is, a highly correlated sample set, so it needs a lot of samples to get any reasonable estimation of the expected distribution attributes.
- Although the Markov chain will eventually converge to the desired distribution, the initial sample may follow a very different distribution, especially when the starting point is in the low-density region. Therefore, a burn out period is usually needed, during which the initial number of samples (for example, the first 1000 or so) is discarded.

On the other hand, most simple rejection sampling methods are affected by dimensions, that is, the probability of rejection is exponentially increased with the increase of dimensions. There is no such problem in metropolis Hastings and other MCMC methods, so MCMC method is often the only available solution when there are many dimensions of the distribution to be sampled. Therefore, MCMC is often the preferred method to generate samples in hierarchical Bayesian models and other high-dimensional statistical models used in many disciplines.

vi. Beta Simulation 10

Given a target density *f*, we build a Markov kernel *K* with stationary distribution *f* and then generate a Markov chain X_t using this kernel so that the limiting distribution of X_t is *f* and integrals can be approximated according to the Ergodic Theorem.
Compared with an algorithm like B-M Algorithm that directly generates independent samples from a distribution, Metropolis–Hastings and other MCMC algorithms have a number of disadvantages:

- The samples are related. Even if they do follow P (x) correctly in the long run, a set of adjacent samples will be related to each other and cannot reflect the distribution correctly. This means that if we want a group of independent samples, we must discard most of the samples and only take every n samples, for a certain value of n (usually determined by examining the autocorrelation between adjacent samples). Autocorrelation can reduce autocorrelation by increasing the jump width (the average size of the jump width, which is related to the variance of the jump distribution), but it also increases the possibility of rejecting the proposed jump. If the jump size is too large or too small, it will lead to a slow hybrid Markov chain, that is, a highly correlated sample set, so it needs a lot of samples to get any reasonable estimation of the expected distribution attributes.
- Although the Markov chain will eventually converge to the desired distribution, the initial sample may follow a very different distribution, especially when the starting point is in the low-density region. Therefore, a burn out period is usually needed, during which the initial number of samples (for example, the first 1000 or so) is discarded.

On the other hand, most simple rejection sampling methods are affected by dimensions, that is, the probability of rejection is exponentially increased with the increase of dimensions. There is no such problem in metropolis Hastings and other MCMC methods, so MCMC method is often the only available solution when there are many dimensions of the distribution to be sampled. Therefore, MCMC is often the preferred method to generate samples in hierarchical Bayesian models and other high-dimensional statistical models used in many disciplines.

vi. Beta Simulation

Given a target density *f*, we build a Markov kernel *K* with stationary distribution *f* and then generate a Markov chain X_t using this kernel so that the limiting distribution of X_t is *f* and integrals can be approximated according to the Ergodic Theorem.

The **Metropolis-Hastings algorithm** is a general purpose MCMC method for approximating a *f*. Given the target density *f* and a conditional density $q(y | x)$ that is easy to simulate from. In addition, q can be almost arbitrary in that the only theoretical requirements are that the ration $\frac{f(y)}{q(y | x)}$ is known up to a constant *independent* of x and that $q(\cdot | x)$ has enough dispertion to lead to an exploration of the entire support of *f*

We can rely on the feature of Metropolis-Hatings algorithm that for every given *q*, we can then construct a Metropolis-Hastings kernel such that *f* is its

 on.

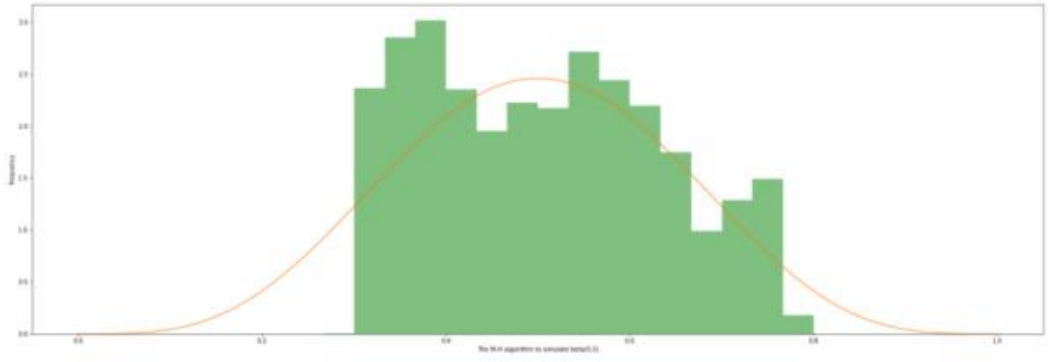
Algorithm

We have $p | X=x, \sim beta(exp(x),a,b)$.

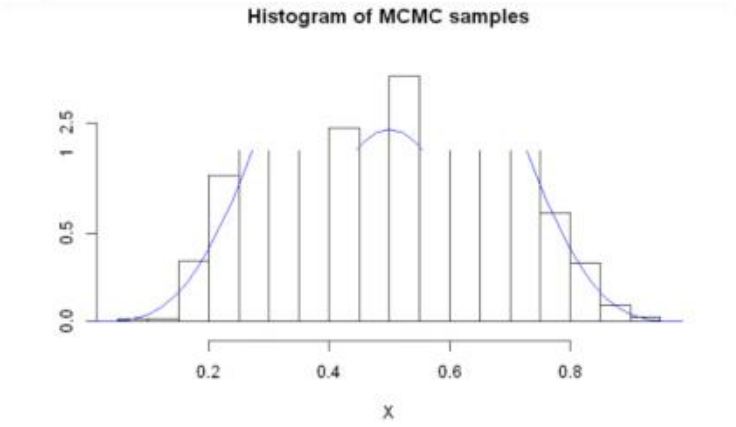
- update X from the $\operatorname{Beta}(exp(x), a,b)$ distribution.
  - sample $p^{*} \sim f_{p}\left(p | p^{(s)}\right)$
  - compute $r=\frac{f_{0}\left(p^{*}, N^{(s)}\right) }{ f_{0}\left(p^{(s)}, N^{(s)}\right)}$
  - set $p^{(s+1)}$ to p^{*} or $p^{(s)}$ with probability $\min (1, r)$ and $\max (0,1-r)$

In [438]:
```
N=5000
X=np.zeros(N)
X[0]=beta.rvs(5,5)
def beta_dist_prob(x,y):
    y = beta.pdf(x,5,5)*expon.pdf(y,0,1/x)
    return y

for i in range(0,N-1):
    Y=expon.rvs(0,1/X[i])
    a=beta_dist_prob(X[i],Y)
    rho=(beta_dist_prob(Y,X[i]))*(beta_dist_prob(Y,X[i]))
    if 0.1<rho:
        X[i+1]=Y
    else:
        X[i+1]=X[i]
fig = plt.gcf()
```

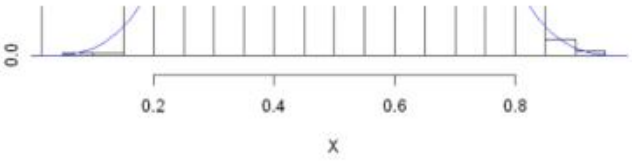
```
fig.set_size_inches(30, 10)
plt.hist(X, 30,range=[0,1], density=1, facecolor='green', alpha=0.5)
plt.ylabel("frequency")
plt.xlabel("X")
plt.xlabel("The M-H algorithm to simulate beta(5,5)")
x = np.linspace(0, 1, 100)
pdf = beta(5, 5).pdf(x)
plt.plot(x, pdf)
plt.show()
```



same simulation using R

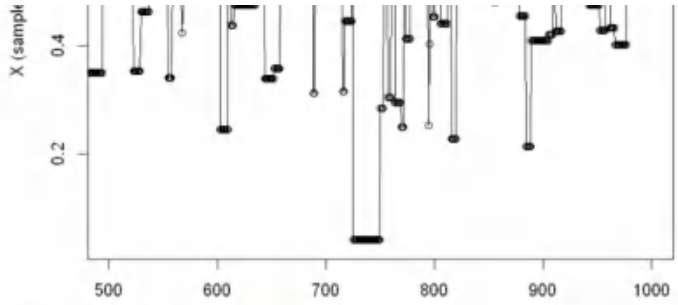



```
N = 10000
X = numeric(N)
X[1] = rbeta(n = 1, shape1 = 5, shape2 = 5) ## initial value
for(i in 1:N){
  Y = rexp(n = 1, rate = X[i])
  rho = (dbeta(x = Y, 5, 5) * dexp(x = X[i], rate = Y)) /
    (dbeta(x = X[i], 5, 5) * dexp(x = Y, rate = X[i]))
  print( Y )
  if(runif(1) < rho){
    X[i+1] = Y
  } else{
    X[i+1] = X[i]
  }

}
```




```
N = 10000
X = numeric(N)
X[1] = rbeta(n = 1, shape1 = 5, shape2 = 5) ## initial value
for(i in 1:N){
  Y = rexp(n = 1, rate = X[i])
  rho = (dbeta(x = Y, 5, 5) * dexp(x = X[i], rate = Y)) /
    (dbeta(x = X[i], 5, 5) * dexp(x = Y, rate = X[i]))
  print( Y )
  if(runif(1) < rho){
    X[i+1] = Y
  } else{
    X[i+1] = X[i]
  }

}
```

MCMC samples




```
plot(X, type = "o", main = "MCMC samples",
slim = c(500,1000),
xlab = "iterations", ylab = "X (samples obtained)")
```

vii. Normal-Normal Conjugacy

10

After observing $Y=y,$ we can update our prior uncertainty for θ using Bayes' rule. Because we are interested in the posterior distribution of $\theta,$ any terms not depending on θ can be treated as part of the normalizing constant. Thus, $$ f_{\theta | Y}(\theta | y) \propto f_{Y | \theta}(y | \theta) f_{\theta}(\theta) \propto e^{-\frac{1}{2 \sigma^{2}}(y-\theta)^{2}} e^{-\frac{1}{2+2}(\theta-\mu)^{2}} $$ since we have a quadratic function of θ in the exponent, we recognize the posterior PDF of θ as a Normal PDF. The posterior distribution stays in the Normal family, which tells us that the Normal is the conjugate prior of the Normal. In fact, by completing the square (a rather tedious calculation which we shall omit), we can obtain an explicit formula for the posterior distribution of $\theta:$ $$ \theta | Y=y \sim \mathcal{N}\left(\frac{\frac{1}{\sigma^{2}}}{\frac{1}{\sigma^{2}}+\frac{1}{\tau^{2}}} y+\frac{\frac{1}{\tau^{2}}}{\frac{1}{\sigma^{2}}+\frac{1}{\tau^{2}}} \mu, \frac{1}{\frac{1}{\sigma^{2}}+\frac{1}{\tau^{2}}}\right) $$ Let's try to make sense of this formula.

- It says that the posterior mean of $\theta, E(\theta | Y=y)$, is a weighted average of the prior mean μ and the observed data $y .$ The weights are determined by how certain we are about θ before getting the data and how precisely the data are measured. If we are already very sure about θ even before getting the data, then τ^{2} will be small and $1 / \tau^{2}$ will be large, which will give a lot of weight to the prior mean μ On the other hand, if the data are very precise, then σ^{2} will be small and $1 / \sigma^{2}$ will be large, which will give a lot of weight to the data y
- For the posterior variance, if we define precision to be the reciprocal of variance, then the result simply says that the posterior precision of θ is the sum of the prior precision $1 / \tau^{2}$ and the data precision $1 / \sigma^{2}$

We can do this by simulating from the posterior distribution of $\theta,$ using the Metropolis-Hastings algorithm to construct a Markov chain whose stationary distribution is $f_{\theta | Y}(\theta | y) .$ The same method can also be applied to a wide variety of distributions that are far more complicated than the Normal to work with analytically. A Metropolis-Hastings algorithm for generating $\theta_{0}, \theta_{1}, \ldots .$ is as follows.

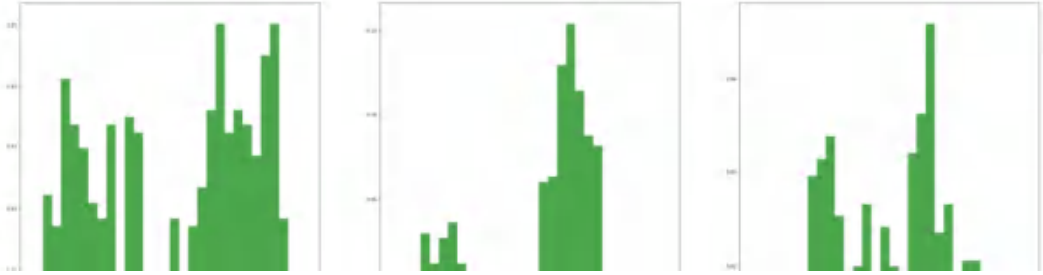
M-H Algorithm

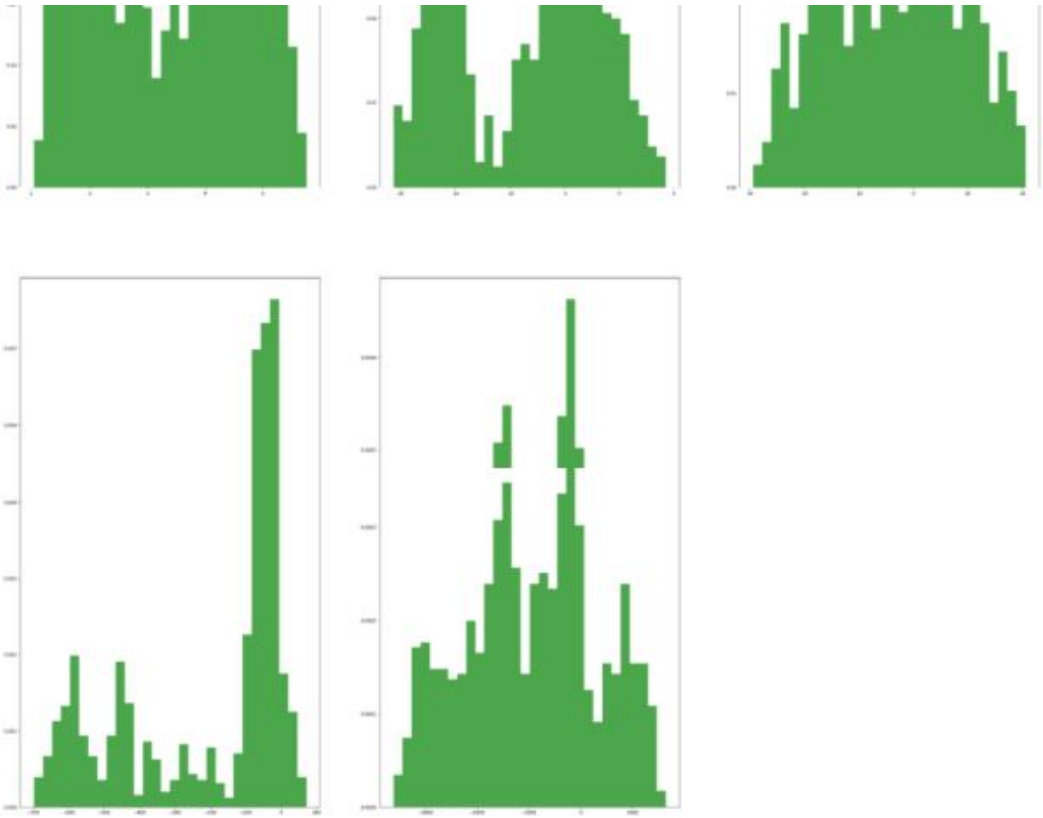
1. If $\theta_{n}=x,$ propose a new state $x^{\prime}$ according to some transition rule. One way to do this in a continuous state space is to generate a Normal r.v. ϵ_{n} with mean 0 and add it onto the current state to get the proposed state: in other words, we generate $\epsilon_{n} \sim \mathcal{N}\left(0, d^{2}\right)$ for some constant $d,$ and then set $x^{\prime}=x+\epsilon_{n} .$ This is the analog of a transition matrix for a continuous state space. The only additional detail is deciding $d .$ in practice, we try to choose a moderate value that is neither too large nor too small.
2. The acceptance probability is $$ a\left(x, x^{\prime}\right)=\min \left(\frac{s\left(x^{\prime}\right) p\left(x^{\prime}, x\right)}{s(x) p\left(x, x^{\prime}\right)}, 1\right) $$ where s is the desired stationary PDF (this was a PMF in the discrete case) and $p\left(x, x^{\prime}\right)$ is the probability density of proposing $x^{\prime}$ from x (this was $p_{i j}$ in the discrete case).

In this problem, we want the stationary PDF to be $f_{\theta | Y},$ so we'll use that for $s .$ As for $p\left(x, x^{\prime}\right),$ proposing $x^{\prime}$ from x is the same as having $\epsilon_{n}=x^{\prime}-x$ so we evaluate the PDF of ϵ_{n} at $x^{\prime}-x$ to get $$ p\left(x, x^{\prime}\right)=\frac{1}{\sqrt{2 \pi} d} e^{-\frac{1}{2 d^{2}}\left(x^{\prime}-x\right)^{2}} $$ However, since $p\left(x^{\prime}, x\right)=p\left(x, x^{\prime}\right),$ these terms cancel from the acceptance probability, leaving us with $$ a\left(x, x^{\prime}\right)=\min \left(\frac{f_{\theta | Y}\left(x^{\prime} | y\right)}{f_{\theta | Y}(x | y)}, 1\right) $$ Once again, the normalizing constant cancels in the numerator and denominator of the acceptance probability.

3. Flip a coin that lands Heads with probability $a\left(x, x^{\prime}\right),$ independently of the Markov chain.
4. If the coin lands Heads, accept the proposal and set $\theta_{n+1}=x^{\prime} .$ Otherwise, stay in place and set $\theta_{n+1}=x$

In [401]:
```
y=3
sigma=1
mu=0
tau=2
d=[0.1, 0.5, 1.5, 10, 100]
T=1000
theta=np.zeros(T)
theta[0]=y
fig = plt.gcf()
fig.set_size_inches(40, 40)
for di in d:
    theta=np.zeros(T)
    theta[0]=y
    for i in range(1,1000):
        theta[i]=theta[i-1]+norm.rvs(loc=0,scale=di)
        r=norm.pdf(y,theta[i],sigma)*norm.pdf(theta[i],mu,tau)*(norm.pdf(y,theta[i-1],sigma)*norm.pdf(theta[i-1],mu,tau))
        flip=binom.rvs(1,1,min(r,1))
        if flip==1:
            theta[i]= theta[i]
        else:
            theta[i]= theta[i-1]
    plt.subplot(int("23"+str(d.index(di)+1)))
    plt.hist(theta,30,density=1, facecolor='green', alpha=0.7)
plt.show()
```





Processing math: 100%

Comments on the Normal-Normal Conjugacy

We ran the algorithm for 10^{4} iterations with the settings $Y=3, \mu=0, \sigma^{2}=1$ $\tau^{2}=4,$ and $d=1 .$ Figures show a histogram of the resulting draws from the posterior distribution of $\theta .$ The posterior distribution indeed looks like a Normal curve. We can estimate the posterior mean and variance using the sample mean and sample variance. For the draws we obtained, the sample mean is 2.4 and the sample variance is 0.8. These are in close agreement with the theoretical values: $$ \begin{aligned} E(\theta | Y=3) &=\frac{\frac{1}{\sigma^{2}}}{\frac{1}{\sigma^{2}}+\frac{1}{\tau^{2}}} y+\frac{\frac{1}{\tau^{2}}}{\frac{1}{\sigma^{2}}+\frac{1}{\tau^{2}}} \mu=\frac{1}{1+\frac{1}{4}} \cdot 3+\frac{\frac{1}{4}}{1+\frac{1}{4}} \cdot 0=2.8 \\ \operatorname{Var}(\theta | Y=3) &=\frac{1}{\frac{1}{\sigma^{2}}+\frac{1}{\tau^{2}}}=\frac{1}{1+\frac{1}{4}}=0.8 \end{aligned} $$

Typesetting math: 100%

Discussion on different variables

different values of $y, \mu, \sigma^{2},$ and τ^{2}

ix. Bivariate Standard Normal Distribution 10

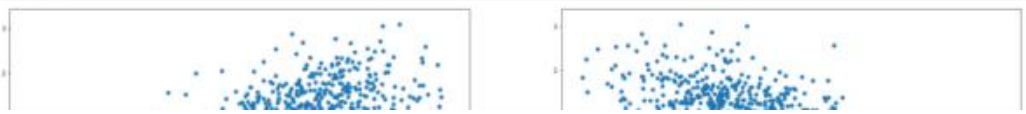
Consider a bivariate standard normal distribution with correlation ρ The bivariate normal distribution has the property that conditional distributions are normal. If (X, Y) has a bivariate standard normal distribution, then the conditional distribution of X given $Y=y$ is normal with mean ρy and variance $1-\rho^{2} .$ Similarly, the conditional distribution of Y given $X=x$ is normal with mean ρx and variance $1-\rho^{2}$

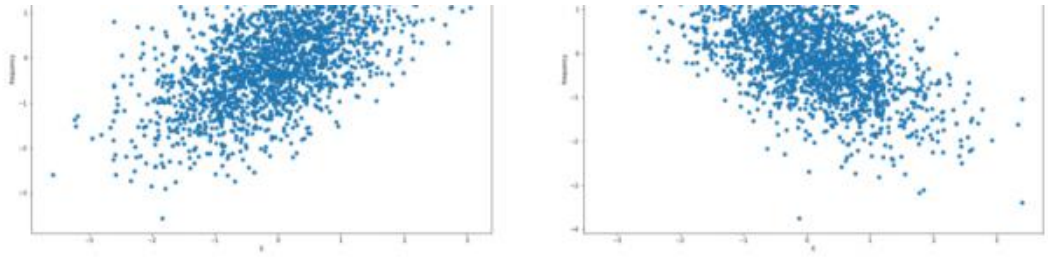
The Gibbs Sampling algorithm

The Gibbs sampler is implemented to simulate (X, Y) from a bivariate standard normal distribution with correlation $\rho .$ At each step of the algorithm, one component of a two-element vector is updated by sampling from its conditional distribution given the other component. Updates switch back and forth. The resulting sequence of bivariate samples converges to the target distribution.

1. Initialize: $\left(x_{0}, y_{0}\right) \leftarrow(0,0)$ $m \leftarrow 1$
2. Generate x_{m} from the conditional distribution of X given $Y=y_{m-1}$. That is, simulate from a normal distribution with mean ρy_{m-1} and variance $1-\rho^{2}$
3. Generate y_{m} from the conditional distribution of Y given $X=x_{m} .$ That is, simulate from a normal distribution with mean ρx_{m} and variance $1-\rho^{2}$
4. $m \leftarrow m+1$
5. Return to Step 2. We simulated a bivariate standard normal distribution with $\rho=0.60,-0.60$ using the Gibbs sampler. The chain was run for 2,000 steps. In R, the output is a 2000×2 matrix.

In [436]:
```
T=2000
rhoo=np.array([0.6,-0.6])
sim=np.zeros((2,2*T))
sdev=np.sqrt(1-np.power(rhoo,2))[0]
fig = plt.gcf()
fig.set_size_inches(30, 10)
for rho in rhoo:
    for i in range(1,T-1):
        sim[0][i]=np.random.normal(rho*sim[1][i-1],sdev)
        sim[1][i]=np.random.normal(rho*sim[0][i],sdev)
    plt.subplot(int("12"+str(np.where(rhoo==rho)[0][0]+1)))
    plt.scatter(sim[0], sim[1])
    plt.ylabel("frequency")
    plt.xlabel("X")

plt.show()
```





x. Chicken-Egg with Unknown Parameters 10

The chicken-egg problem is called chicken or the egg causality dilemma, we suppose a chicken lays a random number of eggs, N, where $N ~ Pois(\lambda)$. Each egg independently hatches with probability p and fails to hatch with probability $q= 1-p$. Let X be the number of eggs that hatch and Y the number that do not hatch, so $X +Y = N$. The joint PMF of X and Y can be seeked for nonnegative intergers i and j.

Basic solution for the problem

We seek the joint PMF P(X = i,Y = j) for nonnegative integers i and j. Conditional on the total number of eggs N, the eggs are independent Bernoulli trials with probability of success p, so by the story of the Binomial, the conditional distributions of X and Y are X|N = n ~ Bin(n,p) and Y|N = n ~ Bin(n, q). Since our lives would be easier if only we knew the total number of eggs, let's use wishful thinking: condition on N and apply the law of total probability. This gives $P(X=i,Y=i)=\Sigma^\infty_{n=0}P(X=i,Y=j|N=n)P(N=n)$, For all possible values of n, fix i and j, we have $\begin{cases}P(X=i,Y=j|N=n)=0\ \ \ \ \ n\neq i+j \\P(X=i,Y=j)=P(X=i,Y=j|N=i+j)\ n=i+j\end{cases}$. Condition on $N=i+j$, the events $X=i$ and $Y=j$ are exactly teh same event, so we have to reduce it by keeping $X=i$, the rest is a matter of plugging in the Bernomial PMF to get $P(X=i|N=i+j)$. Thus, we have $ P(X=i, Y=j) =P(X=i | N=i+j) P(N=i+j)=\left(\begin{array}{c}i+j \\ i\end{array}\right) p^{i} q^{j} \cdot \frac{e^{-\lambda} \lambda^{i+j}}{(i+j) !} =\frac{e^{-\lambda p}(\lambda p)^{i}}{i !} \cdot \frac{e^{-\lambda q}(\lambda q)^{j}}{j !} $ so, we have the basic setting that i.i.d variable's marginal PMFs $X \sim \operatorname{Pois}(\lambda q)$ and $Y \sim \operatorname{Pois}(\lambda q)$

We can rewrithe the PMF as: $f(p | X=x) \propto P(X=x | p) f(p) \propto e^{-\lambda p}(\lambda p)^{x} p^{a-1} q^{b-1}$. By M-H and Gibbs, we can fix the varibles and then make the best guess.

Algorithm for Metropolis–Hastings sampling

We have $p | X=x, N=n \sim \operatorname{Beta}(x+a, n-x+b)$.

1. update p from the $\operatorname{Beta}(x+a, n-x+b)$ distribution.
   - sample $p^{*} \sim f_{p}\left(p | p^{(s)}\right)$
   - compute $r=\frac{f_{0}\left(p^{*}, N^{(s)}\right) }{ f_{0}\left(p^{(s)}, N^{(s)}\right)}$
   - set $p^{(s+1)}$ to p^{*} or $p^{(s)}$ with probability $\min (1, r)$ and $\max (0,1-r)$
2. update N from the $\operatorname{Pois}(\lambda(1-p))$ distribution
   - sample $N^{*} \sim f_{v}\left(N | N^{(s)}\right)$
   - compute $r=\frac{f_{0}\left(p^{(s+1)}, N^{*}\right) }{ f_{0}\left(p^{(s+1)}, N^{(s)}\right)}$
   - set $N^{(s+1)}$ to N^{*} or $N^{(s)}$ with probability $\min (1, r)$ and $\max (0,1-r)$

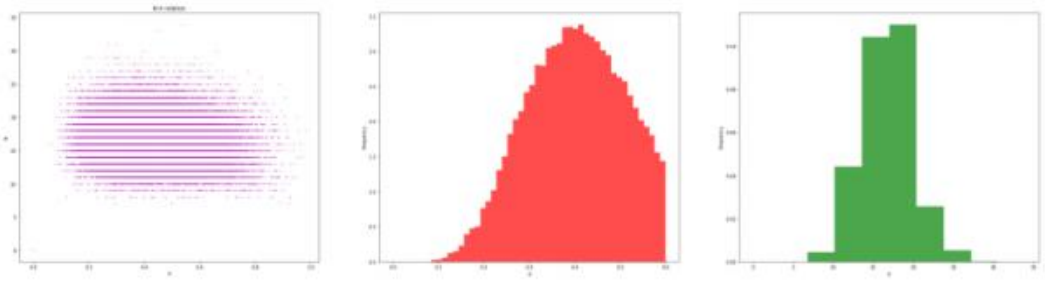
In [109]:
```
def beta_dist_prob(a,b,x,N):
    return(beta.rvs(x+a,N-x+b))

def poisson_dist_prob(lambd,x,p):
    return(poisson.rvs(lambd*(1-p))+x)

p = np.zeros(50000)
N = np.zeros(50000)
a = 1
b = 1
sigma = 1
lambd=10
x=7

(p[0],N[0])=(0.5,2*x)

for t in range(1,50000-1):
    p_star = beta.rvs(x+a,N[t-1]-x+b,size=1)
    alpha = min(1, (poisson_dist_prob(lambd,x,p_star[0]) / poisson_dist_prob(lambd,x,p[t - 1])))
    u = random.uniform(0, 1)
    if u < alpha:
        p[t] = p_star[0]
    else:
        p[t] = p[t - 1]
    N_star = poisson.rvs(lambd*(1-p[m-1]),size=1)+x
    alpha = min(1, (beta_dist_prob(a,b,x,N_star[0]) / beta_dist_prob(a,b,x,N[t - 1])))
    u = random.uniform(0, 1)
    if u < alpha:
        N[t] = N_star[0]
    else:
        N[t+1] = N[t - 1]
```

In [109]:
```
def beta_dist_prob(a,b,x,N):
    return(beta.rvs(x+a,N-x+b))

def poisson_dist_prob(lambd,x,p):
    return(poisson.rvs(lambd*(1-p))+x)

p = np.zeros(50000)
N = np.zeros(50000)
a = 1
b = 1
sigma = 1
lambd=10
x=7

(p[0],N[0])=(0.5,2*x)

for t in range(1,50000-1):
    p_star = beta.rvs(x+a,N[t-1]-x+b,size=1)
    alpha = min(1, (poisson_dist_prob(lambd,x,p_star[0]) / poisson_dist_prob(lambd,x,p[t - 1])))
    u = random.uniform(0, 1)
    if u < alpha:
        p[t] = p_star[0]
    else:
        p[t] = p[t - 1]
```

```
    alpha = min(1, (beta_dist_prob(a,b,x,N_star[0]) / beta_dist_prob(a,b,x,N[t - 1])))
    u = random.uniform(0, 1)
    if u < alpha:
        N[t] = N_star[0]
    else:
        N[t] = N[t - 1]
```



```
fig.set_size_inches(40, 10)
plt.subplot(131)
plt.scatter(p,N, marker = 'x', color = 'm', label='1', s = 3,alpha=0.5)
plt.xlabel("X")
plt.ylabel("N")
plt.title('N-X relation')
plt.subplot(132)
plt.hist(p, 50, density=1,range=[0,0.6], facecolor='red', alpha=0.7)
plt.ylabel("frequency")
plt.xlabel("p")
plt.subplot(133)
plt.hist(N, 10, density=1, facecolor='green', alpha=0.7)
plt.ylabel("frequency")
plt.xlabel("p")
plt.show()
```



Metropolis–Hastings sampling

M-H sampling completely solves the problem of arbitrary probability distribution of the sample set required using the Monte Carlo method, and is therefore widely used in real production environments.

But in the age of big data, M-H sampling faces two major challenges.

1. Our data features are very numerous, and M-H sampling requires considerable computational time at high dimensions due to the acceptance rate computational equation $\frac{\pi(j) Q(j, i)}{\pi(i) Q(i, j)}$, and algorithm efficiency is very low. At the same time $\alpha(i,j)$ is generally less than 1, sometimes painstakingly calculated but rejected. Is it possible to do so without refusing the transfer?
2. Due to the large feature dimensions, it is often even difficult to find the joint distribution of the feature dimensions of the target, but it is convenient to find the conditional probability distribution among the features. At this point can we just sample conveniently with the conditional probability distribution between dimensions?

Gibbs sampling may be a good solution to the problem above.

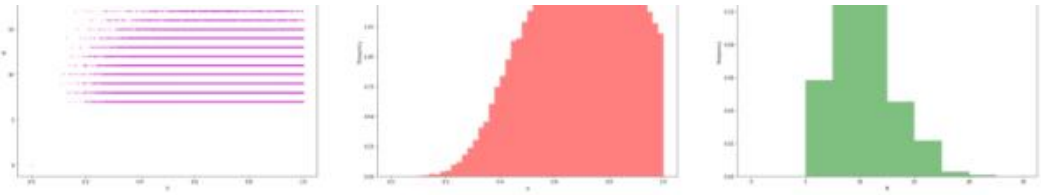
Algorithm for Gibbs sampling

We have $p | X=x, N=n \sim \operatorname{Beta}(x+a, n-x+b)$.

1. Conditional on $N=n$ and $X=x,$ draw a new guess for p from the $\operatorname{Beta}(x+a, n-x+b)$ distribution.
2. Conditional on p and $X=x$, the number of unhatched eggs is $Y \sim \operatorname{Pois}(\lambda(1-p))$ by the chicken-egg story, so we can draw Y from the $\operatorname{Pois}(\lambda(1-p))$ distribution and set the new guess for N to be $N=x+Y$

In [108]:

```
def p_giveN(m,x,N):
    return(np.random.beta(x+a,N[m-1]-x+b))

def N_givenp(m,x,lambd,p):
    return(np.random.poisson(lambd*(1-p[m-1]))+x)

x=7
lambd=10
a=1
b=1

p=np.zeros(50000)
N=np.zeros(50000)

(p[0],N[0])=(0.5,2*x)

for m in range(1,50000-1):
    p[m] = p_giveN(m,x,N)
    N[m] = N_givenp(m,x,lambd,p)

fig = plt.gcf()
fig.set_size_inches(40, 10)
plt.subplot(131)
plt.scatter(p,N, marker = 'x', color = 'm', label='1', s = 3,alpha=0.5)
plt.xlabel("X")
plt.ylabel("N")
plt.title('N-X relation')
plt.subplot(132)
plt.hist(p, 50, density=1, facecolor='red', alpha=0.5)
plt.ylabel("frequency")
plt.xlabel("p")
plt.subplot(133)
plt.hist(N, 10, density=1, facecolor='green', alpha=0.5)
plt.xlabel("N")
plt.ylabel("frequency")
plt.show()
```

Comparison between Metropolis–Hastings and Gibbs Sampling

The Gibbs sampler is a special case of the M-H algorithm. To see this, assume that π is an m-dimensional joint distribution. To avoid excessive notation and simplify the presentation, we just consider one step of the Gibbs sampler when the first component is being updated. Assume that $i = (x_1, x_2,..., x_m)$ is the current state, and $j = (x'_1, x_2,..., x_m)$ is the proposed state. The proposal distribution T is the conditional distribution of X_1 given $X_2,..., X_m$. The acceptance function is $a(i, j) = \frac{\pi_jT_{ji}}{\pi_iT_{ij}}$ We have that $\pi_jT_{ji} = \pi(x'_1, x_2,..., x_m)f_{X_1|X_2,...,X_m} (x_1|x_2,..., x_m) = \pi(x'_1, x_2,..., x_m) \frac {\pi(x_1, x_2,..., x_m)}{ \int \pi(x, x_2,..., x_m) dx) = \pi(x_1, x_2,..., x_m) \frac{ \pi(x'_1, x_2,..., x_m) }{\int \pi(x, x_2,..., x_m) dx) = \pi(x_1, x_2,..., x_m)f_{X_1|X_2,...,X_m} (x'_1|x_2,..., x_m) = \pi_iT_{ij} $

Thus, $a(i, j) = 1$. The proposal state is always accepted. The same is true for all of the components. The algorithm can be implemented by either successively updating each component of the m-dimensional distribution, or by selecting components to update uniformly at random.

More generally, we have the acceptance probability is $\frac{\pi_{(x,y')}p(x,y|x,y')}{\pi_{(x,y)}p(x,y'|x,y)}$=$\frac{p(x,y')p(y'\ is\ chosen\to\ change\ and\ it\ changes\ to\ y|current\ state\ is\ (x,y'))}{p(x,y)p(y\ is\ chosen\to\ change\ and\ it\ changes\ to\ y|current\ state\ is\ (x,y))}$

xi. Three-dimensional Joint Distribution 10

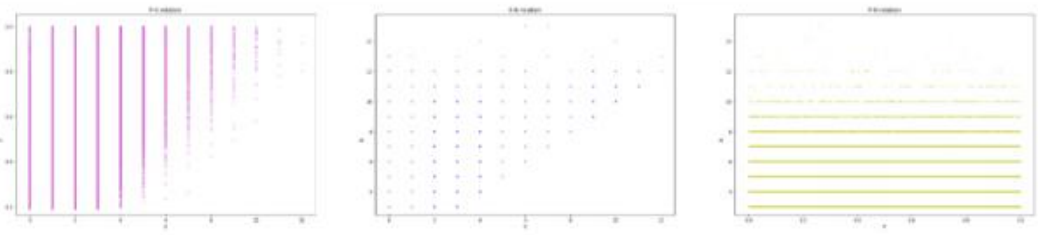
The random variables complies the following joint distributions: $$\pi(x,p,n)\propto{^n_x}p^x(1-p)^{n-x}\frac{4^n}{n!}$$ for $x=0,1,...n, 0<p<1,n=0,1....$ The p vatiable is continuous, x and n are discrete.

The Gibbs sampler requires the variable to be conditionla distribution of the remaining variables that is emulated by the each component. When sampling, the

1. Initialize: $(x0, p0, n0) ← (1, 0.5, 2)$, $m ← 1$
2. Generate x_m from a binomial distribution with parameters $n_{m−1}$ and $p_{m−1}$.
3. Generate p_m from a beta distribution with parameters $x_m + 1$ and $n_{m−1} − x_m + 1$.
4. Let $n_m = z + x_m$, where z is simulated from a Poisson distribution with parameter $4(1 − p_m)$.
5. $m ← m + 1$
6. Return to Step 2.

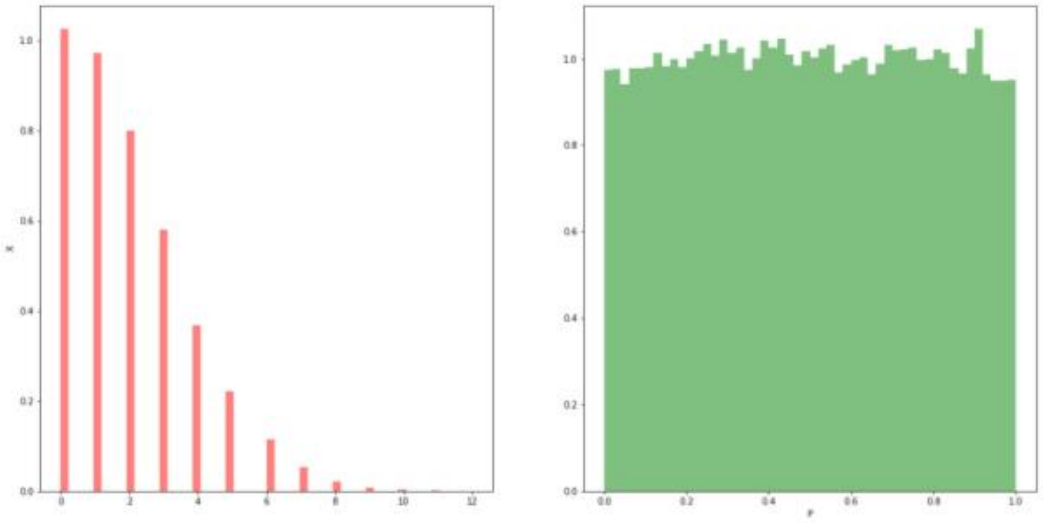
The output of the Gibbs sampler is a sequence of samples $(X_0, P_0,N_0), (X_1, P_1,N_1),(X_2, P_2,N_2),...$

In [54]:
```
def x_givenp(m,n,p):
    return(np.random.binomial(n[m-1],p[m-1]))

def p_givenx(m,n,x):
    return(np.random.beta(x[m]+1,n[m-1]-x[m]+1))

def n_givepx(m,p,x):
    return(np.random.poisson(4*(1-p[m]))+x[m])

N = 50000

n=np.zeros(50000)
x=np.zeros(50000)
p=np.zeros(50000)

(x[0],p[0],n[0])=(1,0.5,2)

for m in range(1,N-1):
    x[m] = x_givenp(m,n,p)
    p[m] = p_givenx(m,n,x)
    n[m] = n_givepx(m,p,x)

fig = plt.gcf()
fig.set_size_inches(40, 10)
    x[m] = x_givenp(m,n,p)
    p[m] = p_givenx(m,n,x)
    n[m] = n_givepx(m,p,x)

fig = plt.gcf()
fig.set_size_inches(40, 10)
plt.subplot(131)
plt.scatter(x,p, marker = 'x', color = 'm', label='1', s = 3,alpha=0.5)
plt.xlabel("X")
plt.ylabel("P")
plt.title('P-X relation')
plt.subplot(132)
plt.scatter(x,n, marker = 'x', color = 'b', label='1', s = 3,alpha=0.5)
plt.xlabel("X")
plt.ylabel("N")
plt.title('X-N relation')
plt.subplot(133)
plt.scatter(p,n, marker = 'x', color = 'y', label='1', s = 3,alpha=0.5)
plt.xlabel("P")
plt.ylabel("N")
plt.title('P-N relation')
plt.show()
```

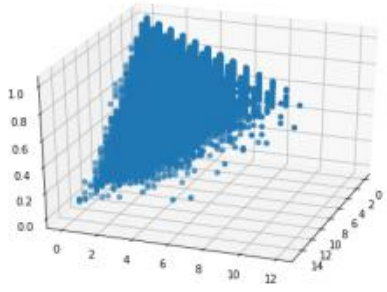


In [57]:
```
fig = plt.gcf()
fig.set_size_inches(20, 10)
plt.subplot(121)
```

```
plt.hist(x, 50, density=1, facecolor='red', alpha=0.5)
plt.ylabel("X")
plt.subplot(122)
plt.hist(p, 50, density=1, facecolor='green', alpha=0.5)
plt.xlabel("P")
plt.show()
```



In [58]:
```
fig = plt.figure()
ax = Axes3D(fig, rect=[0, 0, 1, 1], elev=30, azim=20)
ax.scatter(n, x, p,marker='o')
plt.show()
```



Processing math: 100%

Comments on the result

Due to the advantages of Gibbs sampling at high-dimensional features, Gibbs sampling is currently used in our usual sense of MCMC sampling. Of course Gibbs sampling evolved from M-H sampling, while Gibbs sampling requires data with at least two dimensions, sampling of one-dimensional probability distributions is not possible with Gibbs sampling, at this time M-H sampling still holds.

The Gibbs sampler is remarkably versatile, and can be applied to a large variety of complex multidimensional problems.

xii. Markov Chain Monte Carlo for Wireless Networks 10

Discrete Time - Discrete State - MC

Let's define the state as the binary sequence ($a_1,a_2,a_3...a_{24}$), for every node, once the maximum independent set contains a_n, then, it'll be set to 1.

Basic Property

1. It is irreducible, also positive recurrent. Because all the node can be reached within 24 steps.
2. From an initial state, we randomly propose a new state by flipping a bit uniformly.
   - If the new bit is $0,$ then it will never cause an interference.
   - Or we apply the metropolis-hasting algorithm.

MCMC Algorithm

1. Initialize ($a_1,a_2,a_3...a_{24}$)=$(0, \ldots, 0)$
2. Uniformly choose the state, if the new bit is $0,$ set it and go back to step 1 .
3. Take the acceptance function as $a(i,j)=\min \{1, p=\exp \left(\beta *\left[\sum\ of\ new-sum\ of\ old\right]\right)\}$, β is the variable.
4. - If it sets true, accept the new state and go back to step 1 .
   - Or ignore the new state and go back to step 1.

Typesetting math: 100%

In [591]:
```
beta=np.arange(0.01,10,0.01)

def valid(arr):
    count=0
    for a in arr:
        if a==1:
            count+=1
    return count
def DT_DS_MC(beta):
    history=[]
    state=[0 for i in range(24)]
    for i in range(50000):
        history.append(valid(state))
        #initialization
```

```
        current=-1
        les=999999999

        #start simulation
        for temp_current in range(m):
            if state[temp_current]==0:
                temp=[i for i in state]
                temp[temp_current]=temp[temp_current]^1
                if valid(temp) is not None:
                    flip=np.random.exponential(1/beta)
                    if flip<les:
                        current=temp_current
                        les=flip
            else:
                flip=np.random.exponential(1/beta)
                if flip<les:
                    current=temp_current
                    les=flip
        temp=[i for i in state]
        temp[current]=temp[current]^1
    return np.array(history)
```

In [593]:
```
hist=DT_DS_MC(10)
hist
```

Out[593]:
```
array([ 0,  1,  2, ..., 11, 10,  9])
```

Comments on DT-DS-MV

As shown in the graph, when $\beta=10$ it converges fine and the final result is $9.$

Proof in theory

Determining the size of a maximum independent set of a graph G denoted by $\alpha(G),$ is an NP-hard problem. Therefore many attempts are made to find upper and lower bounds, or exact values of $\alpha(G)$ for special classes of graphs.

This paper is aimed toward studying this problem for the class of generalized Petersen graphs. We find new upper and lower bounds and some exact values for $\alpha(P(n, k)) .$ With a computer program we have obtained exact values for each $n<78 .$ It is conjectured that the size of the minimum vertex cover, $\beta(P(n, k)),$ is less than or equal to $n+\left\lceil\frac{n}{5}\right\rceil,$ for all n and k with $n>2 k$. We prove this conjecture for some cases. In particular, we show that if $n>3 k$ the conjecture is valid. We checked the conjecture with our table for $n<78$ and it had no inconsistency. Finally, we show that for every fixed $k, \alpha(P(n, k))$ can be computed using an algorithm with running time $O(n)$

The upper bound is proved in [6]

Proof Let $S_0 \in \mathcal{S}_{\min}$. We consider two cases.
Case 1: $f(S_0) = 0$.
In this case $T_1(S_0) = \emptyset$. So $|I_t \cap S_0| \leq 2k-1$ for any $1 \leq t \leq n$. If we add all of these n inequalities we get:

$$\sum_{t=1}^{n} |I_t \cap S_0| \leq (2k-1)n. \tag{1}$$

On the other hand $\sum_{t=1}^{n} |I_t \cap S_0| = 2k|S_0|$, since every element of S_0 is

$$2k|S_0| \leq (2k-1)n \longrightarrow \alpha(P(n,k)) = |S_0| \leq \frac{2k-1}{2k}n.$$

Case 2: $f(S_0) > 0$.
In this case $T_1(S_0) \neq \emptyset$. Similar to the inequality (1) we have:

$$2k|S_0| - \sum_{t=1}^{n} |I_t \cap S_0| \leq (2k-1)n + |T_1(S_0)| - |T_3(S_0)|.$$

So, to prove the theorem, it suffices to show that there exists $S_0 \in \mathcal{S}_{\min}$ such that $|T_1(S_0)| \leq |T_3(S_0)|$.
If we can show that for any $I_r \in T_1(S_0)$, there exists an $I_{r'} \in T_3(S_0)$ so that $I_{r+1}, I_{r+2}, \ldots, I_{r'} \notin T_1(S_0)$, then it follows that $|T_1(S_0)| \leq |T_3(S_0)|$.

On the contrary, suppose that there exists $I_t \in T_1(S_0)$ in such a way that in the sequence $I_{t+1}, I_{t+2}, \ldots$ before we see an element of $T_3(S_0)$, we see an element of $T_1(S_0)$. Without loss of generality we can assume that $t-1$. By Lemma 1, $(I_1 \cap S_0)$ is of the form depicted in Figure 2.
Since $I_1 \in T_1(S_0)$, by Corollary 1, $I_2, I_3, \ldots, I_{2k+1} \notin T_1(S_0)$. Based on

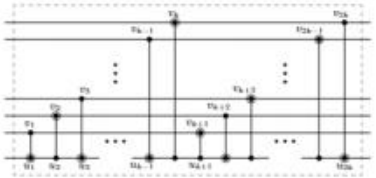


Figure 2: I_1 as a Type 1 segment.

our assumption, $I_2, I_3, \ldots, I_{2k+1} \in T_2(S_0)$. In particular, $I_{2k+1} \in T_2(S_0)$. Since $I_1 \in T_1(S_0)$, by Lemma 2 we have $u_{2k+1}, v_{2k+1} \notin S_0$. On the other

Figure 2: I_1 as a Type 1 segment.

our assumption, $I_2, I_3, \ldots, I_{2k+1} \in T_2(S_0)$. In particular, $I_{2k+1} \in T_2(S_0)$. Since $I_1 \in T_1(S_0)$, by Lemma 2 we have $u_{2k+1}, v_{2k+1} \notin S_0$. On the other hand, we know that S_0 must have one vertex from each edge $u_i v_i$ where $2k+2 \leq i \leq 4k$. Since $2k+2 \leq 2k+3 \leq 4k$, either u_{2k+3} or $v_{2k+3} \in S_0$. But notice that v_{2k+3} is adjacent to v_{k+3} which is in S_0, for $k > 2$. Thus, $v_{2k+3} \notin S_0$ and u_{2k+3} must be in S_0. This means that $u_{2k+2} \notin S_0$. Now, define $S_1 := (S_0 \setminus \{u_{2k}\}) \cup \{u_{2k+1}\}$. One can easily see that $S_1 \in \mathcal{S}$. Based on the choice of $S_0 \in \mathcal{S}_{\min}$, $f(S_0) \leq f(S_1)$. Therefore, there must be an index $2 \leq r \leq n$ so that $I_r \in T_1(S_1) \setminus T_1(S_0)$. Since S_0 and S_1 agree on every element except u_{2k} and u_{2k+1}, the only candidate for r is $r = 2k+1$. So $I_{2k+1} \in T_1(S_1)$ and $I_{2k+1} \notin T_1(S_0)$. Moreover $I_1 \in T_1(S_0)$ and $I_1 \notin T_1(S_1)$. Thus $f(S_0) = f(S_1)$. By Proposition 1, $S_1 \in \mathcal{S}_{\min}$. Notice that if any of $I_{2k+2}, I_{2k+3}, \ldots, I_n$ are of Type i with respect to S_1, they are of Type i with respect to S_0, as well. So, in the sequence $I_{2k+2}, I_{2k+3}, \ldots, I_n$, any Type 3 segment with respect to S_1 appears after an element of Type 1 with respect to S_1. Since $I_{2k+1} \in T_1(S_1)$ by

Corollary 11, $I_{2k+2}, I_{2k+3}, \ldots, I_{4k+1} \notin T_1(S_1)$. Then from our assumption $I_{2k+2}, I_{2k+3}, \ldots, I_{4k+1} \in T_2(S_1)$.

This means that the same argument can be applied to S_1 and if we define $S_2 := (S_1 \setminus \{u_{2k}\}) \cup \{u_{4k+1}\}$, then $S_2 \in \mathcal{S}_{\min}$. If we consecutively repeat this argument for $S_1, S_2, S_3, \ldots, S_m$ where $m = \lfloor \frac{n}{2k} \rfloor$ and $S_i := (S_{i-1} \setminus \{u_{2ik}\}) \cup \{u_{2ki+1}\}$, then we observe that $S_i \in \mathcal{S}_{\min}$ and $I_{2ki+1} \in T_1(S_i)$ for $i = 1, 2, \ldots, m$, and none of $I_2, I_3, \ldots, I_{2k(m+1)}, I_{2k(m+1)+1}$ are of Type 1 with respect to S_0. Also, $I_{2k(i+1)+1}$ for $i = 0, 1, \ldots, m$ are of Special type 2 with respect to S_i. Since S_i and S_0 agree on the $I_{2ki+2}, I_{2ki+3}, \ldots, I_n$, then $I_{2k(i+1)+1}$ for $i = 0, 1, \ldots, m$ are of Special type 2 with respect to S_0.
In other words, if I_1 belongs to $T_1(S_0)$ and the next element of $T_1(S_0)$ appears before the first element of $T_3(S_0)$ in the sequence $I_2, I_3, I_4, \ldots$, then all of $I_{2k+1}, I_{4k+1}, \ldots, I_{2km+1}, I_{2k(m+1)+1}$ are Special type 2 with respect to S_0. In particular, I_{2km+1} is of Special type 2 with respect to S_0.
As $I_{2km+1} \in T_2(S_m)$, by Lemma 9, $u_{2k(m+1)+1}, v_{2k(m+1)+1} \notin S_m$ and since S_m and S_0 agree on the $I_{2km+2}, I_{2km+3}, \ldots, I_n$ we conclude that $u_{2k(m+1)+1}, v_{2k(m+1)+1} \notin S_0$.
Now consider three cases:

- $2k(m+1)+1 \equiv 1 \pmod{n}$:
  Since I_{2km+1} is of Special type 2 with respect to S_0, by Lemma 9, we have $u_{2km+1+2k} = u_1 \notin S_0$. This is a contradiction as we assumed $I_1 \in T_1(S_0)$ and therefore $u_1 \in S_0$.
- $2k(m+1)+1 \not\equiv 0, 1 \pmod{n}$:
  Since $I_1 \in T_1(S_0)$, by Lemma 9, $u_n, v_n \notin S_0$. Also we know that $u_{2k(m+1)+1}, v_{2k(m+1)+1} \notin S_0$. Thus, $I_{2k(m+1)+1}$ is of Type 3 with respect to S_0 and none of $I_2, I_3, \ldots, I_{2k(m+1)}$ are of Type 1 with respect to S_0 which is a contradiction.
- $2k(m+1)+1 \equiv 0 \pmod{n}$:
  I_1 is of Type 1 with respect to S_0 and for every $1 \le i \le m$, $I_{2k(i+1)+1}$ is of Special type 2 with respect to S_0. In particular, I_{2km+1} is of Special type 2 with respect to S_0, and therefore $v_{2km+k+2} = v_{2k(m+1)-k+2} \in S_0$, (See Figure 3). On the other hand, $v_2 \in S_0$ as $I_1 \in T_1(S_0)$, and since $n = 2k(m+1)+1$, v_2 is adjacent to $v_{2k(m+1)-k+2}$. This is a contradiction.

So in all the cases, we get a contradiction which means, after any Type 1 segment I_r, a Type 3 segment $I_{r'}$ will appear before we see another Type 1 segment. This means that $|T_1(S_0)| \le |T_3(S_0)|$ and the theorem follows, as we argued earlier. ■

Typesetting math: 23%

Typesetting math: 55%

Continuous - MC

For the continuous case, we will just modify the above method. We have 2 different situations.

Continuous Time - Discrete State - MC

To begin with, the value function we use in continuous time is the total time it stops at a state (sequence).As we know the CTMC will stops at a state with exponentially amount of time, so more a state is visited, the more time it will stop on this state in total. So the algorithm is almost the same except that before entering next loop, we will add the corresponding stop time to each state which is sampled from exponential distribution.

MCMC Algorithm

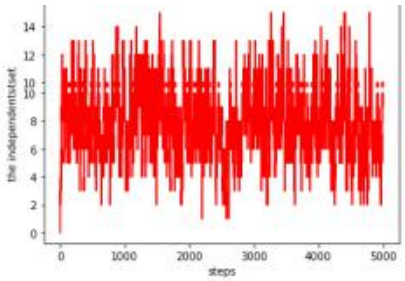
1. Initialize ($a_1,a_2,a_3...a_{24}$)=$(0, \ldots, 0)$
2. Uniformly choose the state, more a state is visited, the more time it will stop on this state in total.
3. Take the acceptance function as $$\alpha_{j} \equiv \lim _{t \rightarrow \infty} P_{i, j}(t) \equiv \lim _{t \rightarrow \infty} P(X(t)=j \mid X(0)=i)=\min \{1, p=\exp \left(\lambda *\left[sum\ of\ new-sum\ of\ old\right]\right)\}$$
4. - If it sets true, accept the new state with the function $$\alpha_{j} \equiv \lim _{t \rightarrow \infty} P_{i, j}(t) \equiv \lim _{t \rightarrow \infty} P(X(t)=j \mid X(0)=i)=\min \{1, p=\exp \left(\mu *\left[sum\ of\ new-sum\ of\ old\right]\right)\}$$ and go back to step 1 .
   - Or ignore the new state and go back to step 1.

So, we can see that CT-DS-MC is similar to DT-DS-MC other than the continuous connected variables. And it has an important concept that is reversibility.

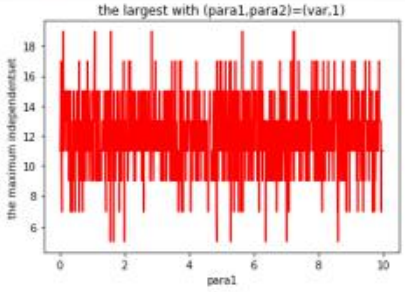
Typesetting math: 100%

In [529]:
```
def valid(arr):
    count=0
    for a in arr:
        if a==1:
            count+=1
    return count
def CT_DS_MC(para1,para2):
    history=[]
    state=[0 for i in range(24)]
    for i in range(5000):
        history.append(valid(state))
        #initialization
        #start simulation
        for temp_current in range(m):
            if state[temp_current]==0:
                temp=[i for i in state]
                temp[temp_current]=temp[temp_current]^1
                if valid(temp) is not None:
                    flip=np.random.exponential(1/para1)
                    if flip<les:
                        current=temp_current
                        les=flip
            else:
                flip=np.random.exponential(1/para2)
                if flip<les:
                    current=temp_current
                    les=flip
        temp=[i for i in state]
        temp[current]=temp[current]^1
        state=temp
    return np.array(history)
```

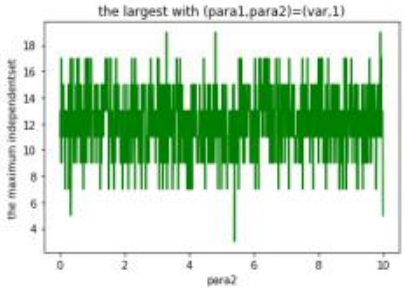
In [527]:
```
hist=CT_DS_MC(1,2)
i=np.arange(5000)
plt.plot(i,hist,'r-')
plt.xlabel("steps")
plt.ylabel("the independentset")
plt.title("the independentset with steps with (para1,para2)=(1,2)")
plt.show()
```



In [531]:
```
hist=[]
para1=np.arange(0.01,10,0.01)
for para in para1:
    hist.append(CT_DS_MC(para,1)[4999])

plt.plot(para1,hist,'r-')
plt.xlabel("para1")
plt.ylabel("the maximum independentset")
plt.title("the largest with (para1,para2)=(var,1)")
plt.show()
```



In [533]:
```
hist=[]
para2=np.arange(0.01,10,0.01)
for para in para2:
    hist.append(CT_DS_MC(1,para)[4999])

plt.plot(para2,hist,'g-')
plt.xlabel("para2")
plt.ylabel("the maximum independentset")
plt.title("the largest with (para1,para2)=(var,1)")
plt.show()
```



Comments on CT-DS-MV

As shown in the graph, when $\lambda =1, \mu=1$ it converges fine and the final result is 9.

Proof in Theory

n \ k	1	2	3	4
5	4	4		
6	6	4		
7	6	5	5	
8	8	6	8	
9	8	7	7	7
10	10	8	10	8
11	10	8	9	9
12	12	9	12	9

Processing math: 100%

Discrete Time - Continuous State - MC

The Metropolis-Hastings algorithm can also be applied in a continuous state space, using PDFs instead of PMFs. This is extremely fiseful in Bayesian inference, where we often want to study the posterior distribution of an unknown parameter. This posterior distribution may be very complicated to work with analytically, and may have an unknown normalizing constant.

Transition probability matrix needs to be replaced by a transition probability function/kernel: $p_{i j}$ becomes $p(x | y)$ which is a probability density function for any given y value. Or a conditional density. Interpretation: $p(x | y) d x$ is the probability of going to x given it is at y now.

We can apply DT-CS-MC in other cases such as Bayesian Analysis.

In statistical problem of Bayesian inference, we are interested in the posterior distribution/density, or the mean of the posterior etc. posterior density $=$ constant $\times$ prior density function $\times$ likelihood function. Example: $X_{i} \sim N\left(\theta, \sigma^{2}\right)$ and the prior on θ is $N\left(\mu, \tau^{2}\right) .$ Here $\sigma^{2}, \tau^{2}, \mu$ are all known.

Algotithm

We can apply the chain with the following rules:

Typesetting math: 91%

At Stage n, suppose a MC takes value y_{n}. We generate the 'Candidate' value Y using the uniform random walk. $Y \sim U$ ni $\left[y_{n}-a, y_{n}+a\right]$. Finally we accept the generated Y value as y_{n+1} if $$ \frac{f(Y)}{f\left(y_{n}\right)}>U(0,1) $$ (where $U(0,1)$ denote another independently generated random variable) otherwise the MC do not move, i.e. $y_{n+1}=y_{n}$

We can show that this transition kernel function, and density function $f(\cdot)$ pair satisfy the 'detailed balance equation'. The transition probability density function is $$ \frac{1}{2 a} I\left[y_{n}-a<Y<y_{n}+a\right] \operatorname{Pr}\left(\frac{f(Y)}{f\left(y_{n}\right)}>U\right) $$ since U is from uniform (0,1), therefore we have $$ =\frac{1}{2 a} I\left[\left|Y-y_{n}\right|<a\right] \min \left\{1, \frac{f(Y)}{f\left(y_{n}\right)}\right\} $$

Comparison between the CT-DS-MC & DT-DS-MC & DT-CS-MC

Comparison between the CT-DS-MC & DT-DS-MC