Determinants of Farm Size Distributions: Project Notes

Matthew J. Baker and Jonathan Conning

Introduction

What determines the equilibrium distribution of operational farm sizes in an economy, and it's evolution over time? These are age old questions that have been studied with different approaches and assumptions with widely varying policy implications.

In the paragraphs and links to other documents below we describe our efforts to build models and code to explore these questions via empirically fitted (or fittable) general equilibrium models of the size distribution of farms. Our illustrative applications have include an empirical application to contemporary US farm agriculture, a replication of a recent 2014 AER paper (https://www.aeaweb.org/articles.php?doi=10.1257/aer.104.6.1667) by Adamopoulos and Restuccia with a broadly similar approach, and models with factor market power which we've used to interpret applications in Russia at the time of the Stolypin reforms (with Aleks Michuda) and the rise and persistence of Latifundia-minifundia bimodal structures in 19th century Chile. As structural models which carefully specify the nature of the underlying production technologies and resource and other constraints, these models lend themselves to rich counter-factual simulations and help to understand how micro-level resource misallocations may lower sectoral and national reductions productivity and output.

We are laying all this out partly in an effort to also define a model framework for exploring the determinants of the size-distribution of farms in the Ukraine.

We've ported most of these models (some of which previously ran in MATLAB or Mathcad) to Ipython notebook format which runs python scripts in between rich text like this. In the section on **Model Implementations** below we've placed links to 'viewer' versions of the Ipython notebooks that implement some of the above applications. Click on them to see the model described in Latex and the code, and output from the python implementation. If you want to edit and interact with the notebook, install Ipython notebook on your machine (we recommend the Anaconda distribution) and then click on the download link in the viewer file (top right corner of browser screen) to save the exectuable file.

Brief historical background and literature review

The economics literature on the determinants of the farm size distribution, and the related question of whether farms are 'too large' or 'too small' has evolved considerably over time. There have been a approaches to the topic, each based on different premises and sometimes hidden or not fully explored assumptions about the underlying technologies and the nature of market failures. Let's briefly review some literature to make this clear, since we are able to adopt elements of each approach in our models.

The development literature approach

The oft cited work of Berry and Cline (1979) as well as theoretical and historical analysis of many others (surveyed in Binswanger, Feder and Deininger, 1995) stressed the importance and historical ubiquity of family-farms. Influenced perhaps by the relatively successful experience of land and tenancy reforms in East Asia and the demand for reforms in land-concentrated Latin America in the 1960s-80s this literature frequently claimed that failures in credit and land lease and land sales markets led to a distorted farm size distribution with inefficiently large estates operating side-by-side with inefficiently smallholder plots. The market failures were sed to lead to persistent divergences in marginal products across farms which opened up room for output-enhancing land redistributions and other policy interventions.

As is well understood, the failure in a single factor market is not enough to cause inefficiency in markets with constant returns to scale production technologies. For example, a failure in the land market or even shutting down that market entirely, need not cause inefficiency if farms can adjust factor proportions and marginal products using markets for labor and other factors. In fact this literature largely accepts as a fact that labor market failures dictate that hired labor is only an imperfect substitute for own or family labor. This gives a competitive advantage to family farms. A typical way to model this is to state that farm management/supervision ability is a non-traded factor, and given that its level is fixed on each farm, this implies diseconomies of scale in the supervision of traded factors. For example if $\widehat{F}(T, L, s)$ is a linear homogenous production function employing land T, labor L and non-tradable farming management/supervision ability s then the production function will exhibit diseconomies of scale in the traded factors T and L for any fixed level of s. This is the type of production function in Lucas (1978) span-of-control models. Consider the simple Cobb-Douglas production function:

$$\widehat{F}(T, L, s) = s^{1-\gamma} \cdot \left[T^{\alpha} L^{1-\alpha}\right]^{\gamma} = s^{1-\gamma} F(T, L)$$

A price-taking firms' optimal size in a competitive economy (i.e. their demand for each factor) will be determined by the owner's non-traded skill level s and the efficient size distribution of firms will be matched to the underlying distribution of non-traded skills.

If we add another factor-market distortion (in addition to the non-tradability of s) then the size distribution of farms may become 'distorted' and inefficient. For example, Eswaran and Kotwal's (1986) paper "Access to Capital and Agrarian Production Organization" (http://are.berkeley.edu/courses/ARE251/2003/papers/Eswaran_Kotwal.pdf) shows that an imperfection on the credit market -- assuming for instance that lenders ration capital (to pay for land and labor inputs) in proportion to farmers' ownership of collarelizable land rather than potential farm profitability -- may give rise to a size distribution with inefficiently large farms coexisting side by side with inefficiently small farms, or worse yet: individuals who in a more efficient economy would have become successful small or medium size farmers may become landless workers.

This literature often made much of the finding of an 'inverse-farm-size' relationship, namely that output per unit land is often observed to decline with land size and small farms use labor and other inputs more intensively. Several papers in this literature derived such a relationship theoretically in effect by having size-biased constraints and in such a context, large farms were inefficiently large, small farms inefficiently small, and land redistribution could often improve allocation.

Other variations are possible. For he simple homothetic production function economy described the efficient scale of production will be matched to each farmer's level of skill s. Imperfections to any one other market, for example in the land market, will in general make it difficult to adjust scale and factor proportions to equalize marginal products across farms even if the labor market functions well. Still another possibility explored theoretically and numerically in one of the notebook links below, are market poser type distortions of the sort Conning (2006) has explored. In these models land concentration (as well as control over credit) can give rise to endogenous market power effects that distort the equilibrium farm size distribution and also give rise to an inefficient inverse farm-size relationship.

Recent 'modern macro' approaches

In recent years'modern macro' economists have turned their attention to similar questions. Klennow and Hsieh (QJE 2009 (https://docs.google.com/viewer?url=http%3A%2F%2Fklenow.com%2FMMTFP.pdf)) and others explore the broad question how misallocation of resources across firm a sector can explain a large part of the observable large productivity differences between sectors and countries.

Adamopoulos and Restuccia (2014 (https://www.aeaweb.org/articles.php?doi=10.1257/aer.104.6.1667)) build a general equilibrium model of the US economy, arguing as above that heterogeneity in the underlying distribution of non-tradable farming ability or skill drives the size distribution of farms. They posit however that farm production is
y in farm managers characterized by a "skill" variable s. The farm production function is specified as non-homothetic CES in form with only capital and land as inputs and with skill s augmenting land:

$$y_a = A\kappa[\theta k^{\rho} + (1-\theta)(sl)^{\rho}]^{\frac{\gamma}{\rho}}$$

As pointed out in the US farm study below this way of modeling things will imply (for the right assumption about the substitution parameterρ and the homogeneity parameter γ) that more skilled farmers operate larger and less capital intensive (less capital per unit land) farms. After making a number of somewhat questionable assumptions to 'close' and 'calibrate' their model to US data the authors take the estimated/calibrated skill distribution from the US economy and apply the same market-clearing model to other economies around the world. This model predicts that most countries ought to have more large farms and less small farms to be more efficient. They conclude that tax and subsidy policies around the world are biased against large farmers (and they in effect estimate the vector of policy distortions). They appear to ignore or be largely oblivious to the earlier literature which argued that in many historical contexts size-bias ran the other way.

We have a number of ways to improve on the the way that A&R have modeled things, including our objections to (1) their assumption of no labor market in agriculture (in effect each laborer runs a farm); (2) some of the questionable ways they've pinned down parameters and otherwise 'closed' the model; (3) the cumbersome way they model and estimate the size distribution of skills; (4) the fact that in their model the initial distribution of assets plays no role and there are no market imperfections other than the anti-large policy distortions.

Some Project pieces

The notebook links below illustrate our approach which allows for elements of the development literature approach (with its focus on the possibility of market failures in the market for credit or land as well as the possibility of market power distortions) as well as (an improved) take on the modern macro approach which allows for skill-biased technologies.

Of course it is hard to build a single model that combines all elements and then separately empirically identify the contribution of each element. But it's useful to explore models that combine these elements in different ways to then adapt to the circumstances at hand. For example, an application to the Ukraine ought to incorporate unique elements of the context (for example the ban on land sales and the huge amount of reverse tenancy).

There are three models below: (1) An estimated model of the US farm size distribution; (2) a quick port of the 'market power' distortions model; and (3) the Adamopolous and Restuccia replication. Models (1) and (3) are estimated on data.

An estimated model of the US farm size distribution

Here we take 2011 data on the size distribution of farms as well as data on capital and labor use from the US Department of Agriculture to estimate a structural model of the US farm size distribution. All the parameters of the production function and the skill distribution are estimated using a variation on the method of moments.

Although we present this model first, it helps to also understand Adamopolus and Restuccia's model, since we developed this model in large degree to improve upon that model, most obviously by incorporating labor into the production function where they had just land and capital.

Click here to see the model in action: An estimated model of the US farm size distribution (http://nbviewer.ipython.org/url/www.sugarsync.com/pf/D21658_69095890_45728099/%3FdirectDownload%3Dtrue)

Market-power distorted farm-size distribution

This notebook demonstrates general equilibrium effects of endogenous market power distortions on the farm size distribution, building on earlier work by Jonathan. In order to highlight the mechanisms at work this notebook shows results for the simple case of a homothetic production technology (where skill augments all factors and not in a land-biased way as in the other two approaches demonstrated above and below) and a simple representation of the skill distribution. It is however easy enough to simply swap in a different production techology (and associated demands) to run this with any type of production technology and run it with a lognormal skill distribution (we have done all of this in earlier MATLAB). Finally, although we are not now estimating the parameters of this model (as in the notebook above and below) it should be easy enough to do so using the calibration 'wrapper' function used above.

Click here to see the model in action: Market power distortions in general equilibrium (http://nbviewer.ipython.org/url/www.sugarsync.com/pf/D21658_69095890_45780185/%3FdirectDownload%3Dtrue)

The Adamopolus and Restuccia Replication

The title says it all. More details in the notebook. Click here to see the model in action: A&R replication (http://nbviewer.ipython.org/urls/www.sugarsync.com/pf/D21658_69095890_45728094/%3FdirectDownload%3Dtrue)

In [ ]:

An Estimated model of the US Farm Size Distribution

The following presents a relatively parsimonious three-factor general equilibrium model of the farm size distribution for the US agricultural economy which is then calibrated/estimated using the method of moments. We model production using a nested three-factor CES production technology that employs land, capital and labor and that is augmented by non-traded farming skill/managerial talent.

The analysis is broadly similar in focus to the recent Adamopoulus and Restuccia's (2014) article The Size Distribution of Farms and International Productivity Differences (https://www.aeaweb.org/articles.php?doi=10.1257/aer.104.6.1667) but with a few improvements: (1) we incorporate labor into the analysis (theirs assumes jus capital and land); (2) we estimate all key parameters in the model to match key variables, and (3) we employ a more parsimonious model to avoid several questionable assumptions.

We use a three-input CES production function capable of generating some key stylized facts about the US farm economy (we demonstrate these facts empirically using USDA data in a section below). Adamopolous/Restuccia also lay them out as important but their modeling is a bit different. These facts for the US economy include:

1. the capital-land and labor-land ratios tend to fall as farm size goes up
2. the capital/labor ratio remains more or less constant at most farm sizes.

We take it that unobserved differences in farming skill or managerial ability determine productivity differences across farms to determine differences in factor inputs and therefore in farm size. This is similar to Lucas' (1978) span-of-control model except that to be consistent with the stylized facts we will work with a non-homothetic production function, one where the unobserved 'skill' or poductivity parameter is thought of as something that enhances land-use management primarily.In the right parameter region this will make it land-augmenting and capital and labor saving, consistent with the stylized facts.

Accordingly, consider the nested CES production function.

$$y = \left[\left(a_l l^{\rho} + a_k k^{\rho} \right)^{\frac{\phi}{\rho}} + s t^{\phi} \right]^{\frac{\gamma}{\phi}}$$

Every would-be farmer has this production function and non-traded skill level s. Faced with labor, capital and land factor prices (w, r, q) land and labor demands will be given by:

$$t = \gamma^{\frac{1}{1-\gamma}} \left[Z^{\frac{\phi}{1-\phi}\frac{1-\rho}{\rho}} + s\left(\frac{s}{q}\right)^{\frac{\phi}{1-\phi}} \right]^{\frac{\gamma-\phi}{\phi(1-\gamma)}} \left(\frac{s}{q}\right)^{\frac{1}{1-\phi}}$$

$$l = \gamma^{\frac{1}{1-\gamma}} \left[Z^{\frac{\phi}{1-\phi}\frac{1-\rho}{\rho}} + s\left(\frac{s}{q}\right)^{\frac{\phi}{1-\phi}} \right]^{\frac{\gamma-\phi}{\phi(1-\gamma)}} Z^{\frac{\phi-\rho}{\rho(1-\phi)}} \left(\frac{a_l}{w}\right)^{\frac{1}{1-\rho}}$$

where $Z = a_l\left(\frac{a_l}{w}\right)^{\frac{\rho}{1-\rho}} + a_k\left(\frac{a_k}{r}\right)^{\frac{\rho}{1-\rho}}$

The equilibrium labor/land ratio is:

$$\frac{l}{t} = \frac{\left[s\left(\frac{s}{q}\right)^{\frac{\rho}{1-\rho}} + a_l\left(\frac{a_l}{w}\right)^{\frac{\rho}{1-\rho}} \right]^{\frac{\phi-\rho}{\rho(1-\phi)}} \left(\frac{a_l}{w}\right)^{\frac{1}{1-\rho}}}{\left(\frac{s}{q}\right)^{\frac{1}{1-\phi}}}$$

This ratio will go down if ϕ is smaller than ρ. The capital-land ratio is given by:

$$\frac{k}{t} = \frac{\left[a_k\left(\frac{a_k}{r}\right)^{\frac{\rho}{1-\rho}} + a_l\left(\frac{a_l}{w}\right)^{\frac{\rho}{1-\rho}} \right]^{\frac{\phi-\rho}{\rho(1-\phi)}} \left(\frac{k}{r}\right)^{\frac{1}{1-\rho}}}{\left(\frac{s}{t}\right)^{\frac{1}{1-\phi}}}$$

Finally, the labor/capital ratio is simply:

$$\frac{l}{k} = \left(\frac{a_l}{a_k}\frac{r}{w}\right)^{\frac{1}{1-\rho}}$$

Let's lay down a few parameter values (note that the ordering of ρ, ϕ, and γ matters) to demonstrate this framework is capable of generating the stylized facts.

In [1]:

```
rho=.3
phi=.5
gam=.7
w=1
r=1
q=1
al=1
ak=1.1
```

A function to compute the capital land ratio as a function of skill level s:

In [2]:
```
def ktrat(s):
    part1=(ak/r)**(1/(1-rho))
    part2=(s/q)**(1/(1-phi))
    part3=(ak*(ak/r)**(1/(1-rho))+al*(al/w)**(1/(1-rho)))**((phi-rho)/
(rho*(1-phi)))
    rat=part3*part1/part2
    return rat
```

Now, a function to compute the land labor ratio:

In [3]:
```
def ltrat(s):
    part1=(al/w)**(1/(1-rho))
    part2=(s/q)**(1/(1-phi))
    part3=(ak*(ak/r)**(1/(1-rho))+al*(al/w)**(1/(1-rho)))**((phi-rho)/
(rho*(1-phi)))
    rat=part3*part1/part2
    return rat
```

Finally, a function to compute the labor capital ratio (which really doesn't have to be a function, but we leave it like this to allow for future flexibility:

In [4]:
```
def lkrat(s):
    part1=(al/w)**(1/(1-rho))
    part2=(ak/w)**(1/(1-rho))
    return (part2/part1)*np.ones(N)
```

Let's just plot these functions and see how they vary with the level of skill (again, this is just variation in s_1, everything else fixed).

In [5]:
```
import numpy as np
import scipy.optimize as so        # scientific functions needed later
```

Here and below we shall assume there are N would be farmers or points in the farm skill distribution.

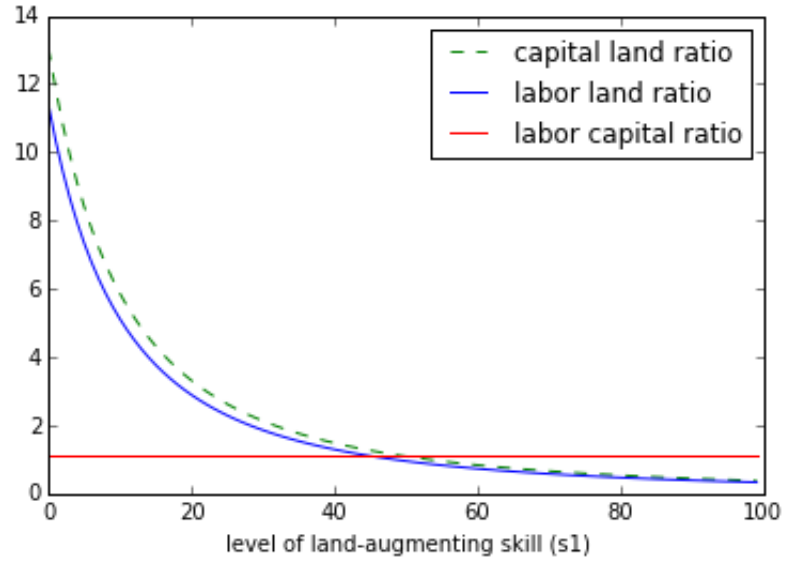
In [6]:
```
N = 100
```

In [7]:
```
KTrat=np.zeros((N,1),float)
LTrat=np.zeros((N,1),float)
LKrat=np.zeros((N,1),float)
si=np.linspace(.51,3,N)               # Not necessarily the distributi
on of skill - we are just plotting ratios as functions of skill
                                      # to see if we get the right rel
ationship.
```

In [8]:
```
KTrat=ktrat(si)             #MATT: no need to loop as already vectoriz
ed - JONATHAN: awesome so that's how this is done!
LTrat=ltrat(si)
LKrat=lkrat(si)
```

In [9]:
```
%pylab inline
import matplotlib.pyplot as plt
plt.plot(KTrat,'--g',label='capital land ratio')
plt.plot(LTrat,'-b',label='labor land ratio')
plt.plot(LKrat,'-r',label='labor capital ratio')
plt.xlabel('level of land-augmenting skill (s1)')
plt.legend()
plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
```



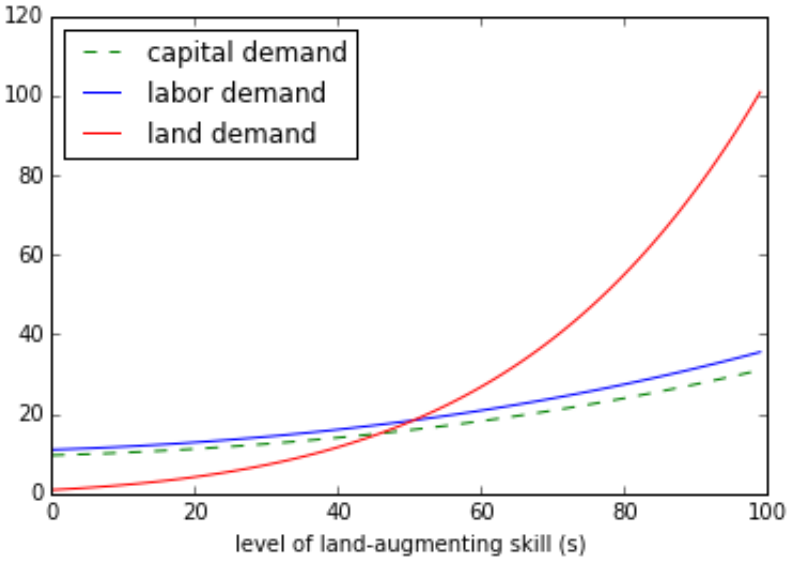
It bears mentioning that this graph captures in a rough way basic empirical facts about agriculture - that labor and capital intensity go down as the size of parcel - which is proportionate to skill - increases. We can also describe how input demands work as functions of skill, which might be even more suggestive:

In [10]:
```
def xDemands(s):
    Z=al*(al/w)**(rho/(1-rho))+ak*(ak/r)**(rho/(1-rho))
    CommonComponent=(Z**((phi/(1-phi))*((1-rho)/rho))+s*(s/q)**(phi/(1
-phi)))
    CommonComponentPower=CommonComponent**((gam-phi)/(phi*(1-gam)))
    ld=gam**(1/(1-gam))*CommonComponentPower*Z**((phi-rho)/(rho*(1-phi
)))*(al/w)**(1/(1-rho))
    kd=gam**(1/(1-gam))*CommonComponentPower*Z**((phi-rho)/(rho*(1-phi
)))*(ak/r)**(1/(1-rho))
    td=gam**(1/(1-gam))*CommonComponentPower*(s/q)**(1/(1-phi))
    return (ld,kd,td)
```

In [11]:
```
LD=np.zeros((N,1),float)
KD=np.zeros((N,1),float)
TD=np.zeros((N,1),float)
LD,KD,TD=xDemands(si)
```

Armed with these demands, we can now plot them and see how they grow as capital augmenting skill goes.

In [12]:
```
plt.plot(LD,'--g',label='capital demand')
plt.plot(KD,'-b',label='labor demand')
plt.plot(TD,'-r',label='land demand')
plt.xlabel('level of land-augmenting skill (s)')
plt.legend(loc='upper left')
plt.show()
```



That seems to be about the way it should look - as farmer skill rises, the farmer demands more of everything, but the demand for land rises faster for land, as opposed to capital and labor. So capital and labor intensity is higher for smaller farms.

Finally, let's look at output per unit input

In [13]:
```
def prodn(s,T,K,L):
    return (( (al*L**rho+ak*K**rho)**phi/rho  + s*T**phi) )**(gam/phi)

YO=prodn(si,TD,KD,LD)
plt.plot(TD,YO/TD,label='output per unit land')
xlabel('land size')
ylabel('output')
plt.title("Inverse-farm size relationship")
```

Out[13]:
```
<matplotlib.text.Text at 0x13983940>
```

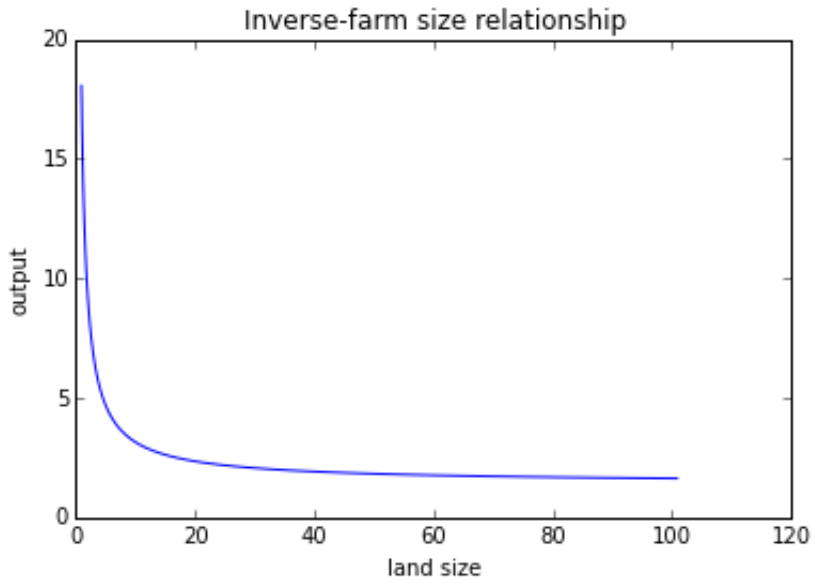


Table 6 of the 2013 USDA's publication "Farm Size and the Organization of U. S. Crop Farming" (http://www.ers.usda.gov/media/1156726/err152.pdf) gives information on the size distribution (in 6 quantiles), for three types of farms - Corn, Soybean, and Wheat farms. Labor and capital input usage is broken into six size classes are Less than 100, 100-249, 250-499,500-999, 1000-1999, and 2000+.

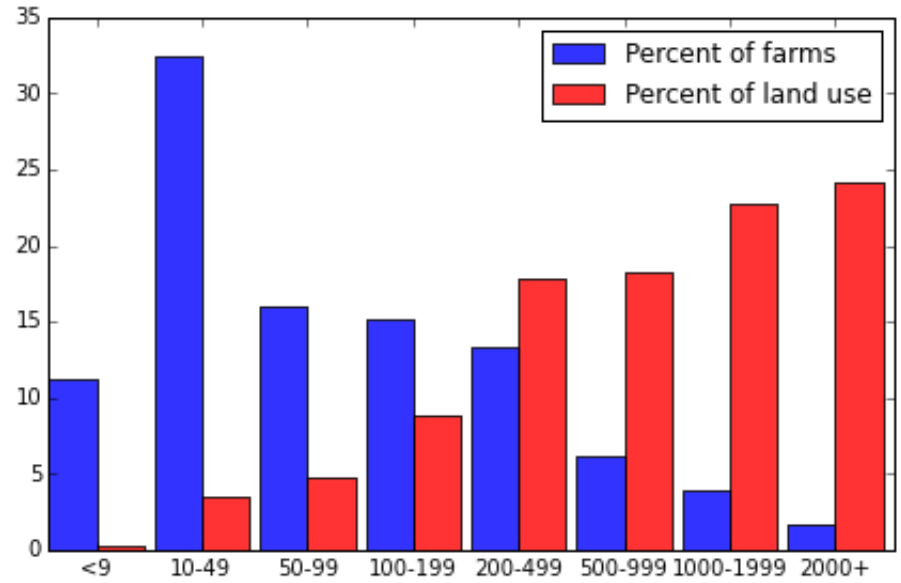
Actual land used is broken into more classes, so we will also input that data. First, the land usage data. The first row is the fraction of firms at each size octile. The second row is the fraction of agricultural land held by those firms. The sizes are Less than 10, 10 to 49, 49-99, 100-249, 250-499,500-999, 1000-1999, and 2000+. Note that the cut points for binning are a little different for labor and capital, where we have 0-100, 100-250, 250-500, 500-999, 1000-1999, and 2000+. So we will also add a term for this:

In [13]:
```
LandOctiles=np.array([[11.2,32.5,16.0,15.1,13.4,6.1,3.9,1.7],[0.2,3.5,
4.7,8.8,17.8,18.2,22.7,24.1]]) #Not really octiles - just eight size g
roups
LandSizeCuts=np.array([10,50,100,250,500,1000,2000])
LandSizeCutsLK=np.array([100,250,500,1000,2000])
```

The bar plot below essentially reproduces the USDA's figure describing this data as

follows:

In [14]:
```
fig, ax = plt.subplots()

index = np.arange(8)
bar_width = 0.45

opacity = 0.8
error_config = {'ecolor': '0.3'}

rects1 = plt.bar(index, np.array(LandOctiles[0,:]), bar_width,
                 alpha=opacity,
                 color='b',
                 label='Percent of farms')

rects2 = plt.bar(index + bar_width, np.array(LandOctiles[1,:]), bar_wi
dth,
                 alpha=opacity,
                 color='r',
                 label='Percent of land use')
ax.set_xticks(index+bar_width)
ax.set_xticklabels( ('<9', '10-49', '50-99', '100-199', '200-499','500
-999','1000-1999','2000+') )


plt.tight_layout()
plt.legend()
plt.show()
```



Here is the labor - to be precise, mean hour per harvested acre, where the first row is corn, the second is soybeans, and the third wheat:

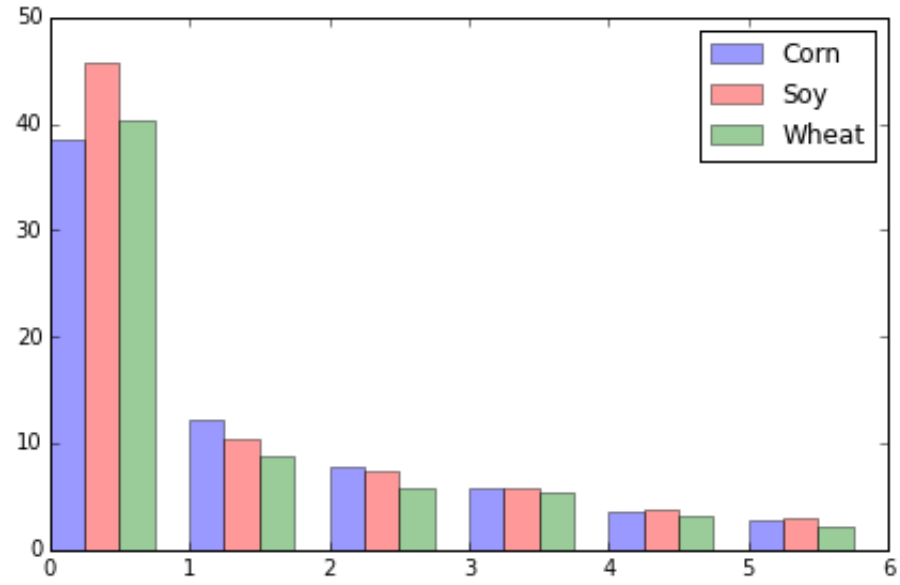
In [15]:
```
LaborSextiles=np.array([[38.6,12.3,7.8,5.7,3.5,2.7],[45.7,10.4,7.3,5.8
,3.8,3.0],[40.4,8.7,5.8,5.3,3.2,2.2]],dtype=float)
```

Next, here is capital:, which is in terms of dollars per harvested acre for each of three crops:

In [16]:
```
CapitalSextiles=np.array([[2532,847,683,568,505,432],[2880,826,640,535
,387,332],[3325,588,396,320,278,242]])
```

In [17]:

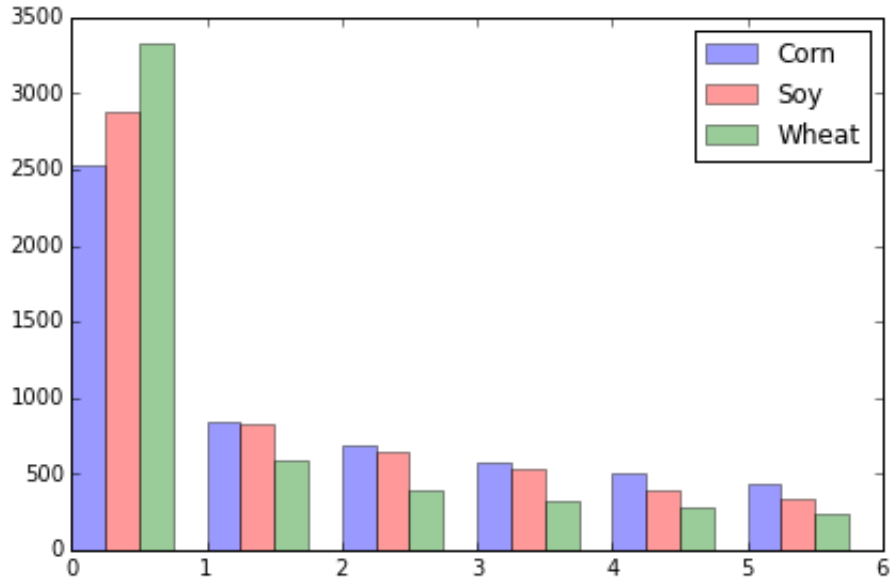
```
fig, ax = plt.subplots()

index = np.arange(6)
bar_width = 0.25

opacity = 0.4
error_config = {'ecolor': '0.3'}

rects1 = plt.bar(index, np.array(LaborSextiles[0,:]), bar_width,
                 alpha=opacity,
                 color='b',
                 label='Corn')

rects2 = plt.bar(index + bar_width, np.array(LaborSextiles[1,:]), bar_
width,
                 alpha=opacity,
                 color='r',
                 label='Soy')

rects3 = plt.bar(index + bar_width*2, np.array(LaborSextiles[2,:]), ba
r_width,
                 alpha=opacity,
                 color='g',
                 label='Wheat')

plt.tight_layout()
plt.legend()
plt.show()
```



The above shows how the skill distribution fades as we move along. Note the close similarity across each crop. Next, let's try capital:

In [18]:

```
fig, ax = plt.subplots()


index = np.arange(6)
bar_width = 0.25

opacity = 0.4
error_config = {'ecolor': '0.3'}

rects1 = plt.bar(index, np.array(CapitalSextiles[0,:]), bar_width,
                 alpha=opacity,
                 color='b',
                 label='Corn')

rects2 = plt.bar(index + bar_width, np.array(CapitalSextiles[1,:]), ba
r_width,
                 alpha=opacity,
                 color='r',
                 label='Soy')

rects3 = plt.bar(index + bar_width*2, np.array(CapitalSextiles[2,:]),
bar_width,
                 alpha=opacity,
                 color='g',
                 label='Wheat')

plt.tight_layout()
plt.legend()
plt.show()
```



Now, a simple empirical exercise we might engage in is to try to get our skill distribution (which means find the parameters of the skill distribution which we shall assume to be lognormal) to generate equilibrium demands that match the sextiles of the capital-land, labor-land, and labor-capital ratios.

In so doing, we first have to decide what things we would like to take as given. Moreover, we have to think about how to translate expenditures on things into actual units (Adamopoulos/Restuccia seem to like to leave these things as free variables in the optimization).

Along these lines, we need a function that takes in a mean and a variance, creates a skill distribution, bins the resulting factor demands, and then sees how closely the empirical distribution matches the actual one.

Before doing this, let's just get a farm rental rate. I'm setting this at 25$, I googled and found a rate of $249 a day for some piece of equipment, so dividing this by 10 working hours in a day gives 25$. Our targets will be:

In [19]:
```
LaborTarget=np.mean(LaborSextiles,axis=0)
CapitalTarget=np.mean(CapitalSextiles,axis=0)
R=25.00
CapitalTarget=CapitalTarget/R
```

In [20]:
```
print CapitalTarget
print LaborTarget
```

```
[ 116.49333333   30.14666667   22.92         18.97333333   15.6
   13.41333333]
[ 41.56666667  10.46666667   6.96666667   5.6          3.5          2.
63333333]
```

Now, let's see if we can calibrate some parameters to match this data, along with the size distribution data on land itself. We will do this to get the basic parameters of the production function along with the skill distribution. So, we will drop those we are trying to match. We will also define stand-alone demand functions, and integrals of these functions:

In [21]:
```
del al,ak,phi,rho,gam                                    # Drop param
eters so we can solve for them instead
```

In [22]:
```
from scipy.stats import norm
from scipy.stats import lognorm
from scipy.integrate import quad
from scipy.optimize import newton
```

In [23]:

```
def LaborDemand(s,al,ak,phi,rho,gam):
    Z=al*(al/w)**(rho/(1-rho))+ak*(ak/r)**(rho/(1-rho))
    CommonComponent=(Z**((phi/(1-phi))*((1-rho)/rho))+s*(s/q)**(phi/(1
-phi)))
    CommonComponentPower=CommonComponent**((gam-phi)/(phi*(1-gam)))
    ld=gam**(1/(1-gam))*CommonComponentPower*Z**((phi-rho)/(rho*(1-phi
)))*(al/w)**(1/(1-rho))
    return ld

def CapitalDemand(s,al,ak,phi,rho,gam):
    Z=al*(al/w)**(rho/(1-rho))+ak*(ak/r)**(rho/(1-rho))
    CommonComponent=(Z**((phi/(1-phi))*((1-rho)/rho))+s*(s/q)**(phi/(1
-phi)))
    CommonComponentPower=CommonComponent**((gam-phi)/(phi*(1-gam)))
    kd=gam**(1/(1-gam))*CommonComponentPower*Z**((phi-rho)/(rho*(1-phi
)))*(ak/r)**(1/(1-rho))
    return kd

def LandDemand(s,al,ak,phi,rho,gam):
    Z=al*(al/w)**(rho/(1-rho))+ak*(ak/r)**(rho/(1-rho))
    CommonComponent=(Z**((phi/(1-phi))*((1-rho)/rho))+s*(s/q)**(phi/(1
-phi)))
    CommonComponentPower=CommonComponent**((gam-phi)/(phi*(1-gam)))
    td=gam**(1/(1-gam))*CommonComponentPower*(s/q)**(1/(1-phi))
    return td

def LaborLandRat(s,al,ak,phi,rho,gam):
    part1=(al/w)**(1/(1-rho))
    part2=(s/q)**(1/(1-phi))
    part3=(ak*(ak/r)**(1/(1-rho))+al*(al/w)**(1/(1-rho)))**((phi-rho)/
(rho*(1-phi)))
    rat=part3*part1/part2
    return rat

def CapitalLandRat(s,al,ak,phi,rho,gam):
    part1=(ak/r)**(1/(1-rho))
    part2=(s/q)**(1/(1-phi))
    part3=(ak*(ak/r)**(1/(1-rho))+al*(al/w)**(1/(1-rho)))**((phi-rho)/
(rho*(1-phi)))
    rat=part3*part1/part2
    return rat
```

In [24]:
```
def intLabor(s,mu,sigma,al,ak,phi,rho,gam):
    val1=LaborLandRat(s,al,ak,phi,rho,gam)
    val2=lognorm.pdf(s,sigma,mu)
    return val1*val2

def intCapital(s,mu,sigma,al,ak,phi,rho,gam):
    val1=CapitalLandRat(s,al,ak,phi,rho,gam)
    val2=lognorm.pdf(s,sigma,mu)
    return val1*val2

def intLand(s,mu,sigma,al,ak,phi,rho,gam):
    val1=LandDemand(s,al,ak,phi,rho,gam)
    val2=lognorm.pdf(s,sigma,mu)
    return val1*val2
```

Just a check to make sure that we can integrate the labor fuunction for some reasonable values:

In [25]:
```
M=.1
S=1.0
AL=1.0
AK=1.0
PHI=.7
RHO=.5
GAM=.8
I1 = quad(intLabor, 0, np.inf, args=(M,S,AL,AK,PHI,RHO,GAM))
I2 = quad(intCapital, 0, np.inf, args=(M,S,AL,AK,PHI,RHO,GAM))
del M,S,AL,AK,PHI,RHO,GAM                                          #Just
 so they aren't laying around - keep that workspace neat!
print I1
print I2
```

```
(32.000462709838644, 1.6126777579773305e-07)
(32.000462709838644, 1.6126777579773305e-07)
```

Next we define an objective function that matches values. We are going to try to completely use the distributional information we have on labor, capital, and land demand. Recall that the labor and capital demand is in sextile form, while the land demand is in quantile form. Let's set up the sextiles and octtiles:

One last bit of theory in pursuit of the idea of developing things in as closed-form as possible. We shall also want to look at the density of farms between the cut points, (0,10),(10,50), etc. Can we form the density directly from the land demand function? Recall that land demand is:

$$t = \gamma^{\frac{1}{1-\gamma}} \left[Z^{\frac{\phi}{1-\phi}\frac{1-\rho}{\rho}} + s\left(\frac{s}{q}\right)^{\frac{\phi}{1-\phi}} \right]^{\frac{\gamma-\phi}{\phi(1-\gamma)}} \left(\frac{s}{q}\right)^{\frac{1}{1-\phi}}$$

What we want is to find the value of s that corresponds with the given threshold or cut point values of $t = 10, 50, 100, 500, 1000, 2000$. Then, we can integrate demand functions between cut points $s(t_1), s(t_2), \ldots s(t_N)$ and then go on from there. To do so, we will have to be able to find a value of s given a value of t. Note we could be more scientific about this given the closed-form (i.e., program in the derivative of the function), but for now this does the job.

In [26]:
```
def findsGivent(s,t,al,ak,phi,rho,gam):
    return t-LandDemand(s,al,ak,phi,rho,gam)

AL=.1
AK=.1
PHI=.5
RHO=.7
GAM=.8
T=1
sCuts=np.zeros((7,1))
newton(findsGivent,1,fprime=None,args=(T,AL,AK,PHI,RHO,GAM))
for i in range(0,7):
    sCuts[i]=newton(findsGivent,LandSizeCuts[i],fprime=None,args=(Land
SizeCuts[i]/10,AL,AK,PHI,RHO,GAM),tol=1e-10,maxiter=1000)
del AL,AK,PHI,RHO,GAM,T
print sCuts
del sCuts
```

```
[[ 1.13325532]
 [ 1.39244988]
 [ 1.52080596]
 [ 1.70818487]
 [ 1.86469125]
 [ 2.03520425]
 [ 2.22100482]]
```

For given parameters our objective tells us how well we match the data, specifically we try to match labor and capital demands for farms of each of the different land size bins as well as get the

In [27]:
```
def objective(X,disp=True):
    mu, sigma = X[0], X[1]
    al, ak =exp(X[2]),exp(X[3])
    gam=exp(X[6])/(1+exp(X[6]))
    phi=exp(X[4])/(1+exp(X[4]))
    rho=exp(X[5])/(1+exp(X[5]))
    sCuts=np.zeros((7,1))
    for i in range(0,7):
        sCuts[i]=newton(findsGivent,LandSizeCuts[i],fprime=None,args=(
LandSizeCuts[i]/10,al,ak,phi,rho,gam),tol=1e-8,maxiter=10000)
    sCutsLK=np.zeros((5,1))
    for i in range(0,5):
        sCutsLK[i]=newton(findsGivent,LandSizeCutsLK[i],fprime=None,ar
```

```
gs=(LandSizeCuts[i]/10,al,ak,phi,rho,gam),tol=1e-8,maxiter=10000)
    interSmeansL=np.zeros((6,1),dtype=float)
    interSmeansK=np.zeros((6,1),dtype=float)
    interOmeansT=np.zeros((8,1),dtype=float)
    densityOT=np.zeros((8,1),dtype=float)
    densityOLK=np.zeros((6,1),dtype=float)
    for i in range(0,8):
 #First compute the lower and upper values of 6 given sextiles
        if i==0:
            interOmeansT[i]=quad(intLand,0,sCuts[i],args=(mu,sigma,al,
ak,phi,rho,gam))[0]
            densityOT[i]=lognorm.cdf(sCuts[i],sigma,mu)
            interOmeansT[i]=interOmeansT[i]/densityOT[i]
        elif i==7:
            interOmeansT[i]=quad(intLand,sCuts[i-1],Inf,args=(mu,sigma
,al,ak,phi,rho,gam))[0]
            densityOT[i]=1-lognorm.cdf(sCuts[i-1],sigma,mu)
            interOmeansT[i]=interOmeansT[i]/densityOT[i]
        else:
            interOmeansT[i]=quad(intLand,sCuts[i-1],sCuts[i],args=(mu,
sigma,al,ak,phi,rho,gam))[0]              # We don't really need this.
..
            densityOT[i]=lognorm.cdf(sCuts[i],sigma,mu)-lognorm.cdf(sC
uts[i-1],sigma,mu)
            interOmeansT[i]=interOmeansT[i]/densityOT[i]
    for i in range(0,6):
        if i==0:
            interSmeansL[i]=quad(intLabor,0,sCutsLK[i],args=(mu,sigma,
al,ak,phi,rho,gam))[0]
            interSmeansK[i]=quad(intCapital,0,sCutsLK[i],args=(mu,sigm
a,al,ak,phi,rho,gam))[0]
            densityOLK[i]=lognorm.cdf(sCutsLK[i],sigma,mu)
            interSmeansL[i]=interSmeansL[i]/densityOLK[i]
            interSmeansK[i]=interSmeansK[i]/densityOLK[i]
        elif i==5:
            interSmeansL[i]=quad(intLabor,sCutsLK[i-1],Inf,args=(mu,si
gma,al,ak,phi,rho,gam))[0]
            interSmeansK[i]=quad(intCapital,sCutsLK[i-1],Inf,args=(mu,
sigma,al,ak,phi,rho,gam))[0]
            densityOLK[i]=1-lognorm.cdf(sCutsLK[i-1],sigma,mu)
            interSmeansL[i]=interSmeansL[i]/densityOLK[i]
            interSmeansK[i]=interSmeansK[i]/densityOLK[i]
        else:
            interSmeansL[i]=quad(intLabor,sCutsLK[i-1],sCutsLK[i],args
=(mu,sigma,al,ak,phi,rho,gam))[0]
            interSmeansK[i]=quad(intCapital,sCutsLK[i-1],sCutsLK[i],ar
gs=(mu,sigma,al,ak,phi,rho,gam))[0]
            densityOLK[i]=lognorm.cdf(sCutsLK[i],sigma,mu)-lognorm.cdf
(sCutsLK[i-1],sigma,mu)
            interSmeansL[i]=interSmeansL[i]/densityOLK[i]
            interSmeansK[i]=interSmeansK[i]/densityOLK[i]
    vals=np.vstack((np.vstack((densityOT,interSmeansL)),interSmeansK))
    targs=np.vstack((np.vstack((np.reshape(LandOctiles[0,:]/100,(8,1))
```

```
,np.reshape(LaborTarget,(6,1)))),np.reshape(CapitalTarget,(6,1))))
    errors=vals-targs
    objVal=np.dot(np.transpose(errors),errors)
    DrawToPrint=np.random.uniform()
    if DrawToPrint>.99:
        print objVal,X
    return objVal
```

Now that we have an objective function defined, let's see if we can actually minimize it with respect to the parameters. If this works, I'll shit my britches. Anyways, here goes:

In [28]:
```
xGuess=-.1,.5,3,3,.5,.3,.7
optResult=so.minimize(objective,xGuess,method='Nelder-Mead',tol=1e-3,o
ptions={'disp': True, 'maxiter':10000, 'maxfev':5000})
```

```
[[  1.05737967e+11]] [-0.1    0.525  3.     3.     0.5    0.3    0.7
  ]
[[  1.83516968e+10]] [-0.10060528  0.50368238  2.91206725  2.8805410
3  0.47733622  0.32110156
  0.70725446]
[[ 662.27702049]] [-0.1532258   0.4773669   1.33123206  1.71603598
0.11866691  0.5225568
  0.7951171 ]
[[ 622.25475854]] [-0.15193387  0.43650419  1.2233608   1.59231618
0.07850013  0.5431108
  0.88244586]
[[ 600.86474417]] [-0.15108092  0.423409    1.1849897   1.56746948
0.0652079   0.55053127
  0.90972397]
[[ 528.17042161]] [-0.13949932  0.37237998  1.05407294  1.42038684
0.01847361  0.58812041
  1.0443045 ]
[[ 519.70049727]] [-0.13478995  0.35680346  1.0115271   1.37049498
0.00912884  0.60120946
  1.08566317]
[[ 507.05578946]] [-0.13418693  0.356031    1.0068153   1.37246563
0.00878892  0.60248808
  1.08802268]
[[ 489.84264733]] [-0.13081571  0.34645645  0.97695892  1.34107166
0.00549386  0.61128963
  1.1131191 ]
[[ 486.78459164]] [-0.12801191  0.33941542  0.95889028  1.31423383
0.00329544  0.61796752
  1.13286803]
[[ 462.47755325]] [-0.1238836   0.32736088  0.91586362  1.28138614
0.0014432   0.62940426
  1.16113103]
[[ 462.07591006]] [-0.1232918   0.32738813  0.91567768  1.28140801
0.00135351  0.63010292
  1.16294392]
[[ 456.74097495]] [ -1.21513843e-01   3.22887352e-01   9.04210923e-0
1   1.26498089e+00
```

```
   9.37141706e-04   6.34454015e-01   1.17452514e+00]
[[ 443.7321528]] [ -1.13981384e-01   3.14478915e-01   8.82067387e-01
   1.23325309e+00
   2.84102977e-04   6.47673474e-01   1.20869776e+00]
[[ 442.86171169]] [ -1.13781651e-01   3.14100587e-01   8.80673781e-0
1   1.23225707e+00
   2.69487047e-04   6.48123018e-01   1.20974924e+00]
[[ 442.35910221]] [ -1.14161689e-01   3.13461657e-01   8.77615975e-0
1   1.23150209e+00
   2.64481299e-04   6.48059312e-01   1.20925116e+00]
[[ 440.70005456]] [ -1.12382500e-01   3.11476930e-01   8.68255684e-0
1   1.22804799e+00
   1.95335651e-04   6.51404139e-01   1.21667682e+00]
[[ 438.70221693]] [ -1.11469889e-01   3.09809638e-01   8.65471734e-0
1   1.22020080e+00
   1.57375450e-04   6.53244485e-01   1.22188204e+00]
[[ 436.28955497]] [ -1.10663522e-01   3.09014945e-01   8.62382079e-0
1   1.21811192e+00
   1.31839276e-04   6.54661176e-01   1.22523348e+00]
[[ 434.25041745]] [ -1.09861143e-01   3.07158736e-01   8.55205038e-0
1   1.21356604e+00
   1.04338174e-04   6.56673118e-01   1.22978240e+00]
[[ 430.72582184]] [ -1.07018425e-01   3.03798804e-01   8.49845074e-0
1   1.19686745e+00
   5.80136181e-05   6.61564861e-01   1.24336221e+00]
[[ 428.4585081]] [ -1.06571707e-01   3.02205130e-01   8.40981681e-01
   1.19562102e+00
   4.73032408e-05   6.63141447e-01   1.24616694e+00]
[[ 749.96906062]] [ -1.06466687e-01   3.02307477e-01   8.41482809e-0
1   1.19570673e+00
   4.71517430e-05   6.63203631e-01   1.24638388e+00]
Optimization terminated successfully.
         Current function value: 428.412623
         Iterations: 1055
         Function evaluations: 1653

C:\Users\Matthew J Baker\Anaconda\lib\site-packages\scipy\integrate\qu
adpack.py:321: IntegrationWarning: The integral is probably divergent,
 or slowly convergent.
  warnings.warn(msg, IntegrationWarning)
```

In [29]:
```
Results=optResult.x
MU=Results[0]
SIGMA=Results[1]
AL=exp(Results[2])
AK=exp(Results[3])
GAMMA=exp(Results[6])/(1+exp(Results[6]))
PHI=exp(Results[4])/(1+exp(Results[4]))
RHO=exp(Results[5])/(1+exp(Results[5]))
print MU,SIGMA
print AL,AK
print GAMMA,PHI,RHO
```

```
-0.106401973072 0.302392771635
2.31991076562 3.30739595444
0.776683352379 0.50001181595 0.659989626167
```

Now, that we have some results, let's see what happens if we look at what they imply. One could do this scientifically (look at the Chi-Square statistic or something from the moment matching problem). But it is probably best to see how close our binned values come to the actual binned values.

First, the size distribution of farms:

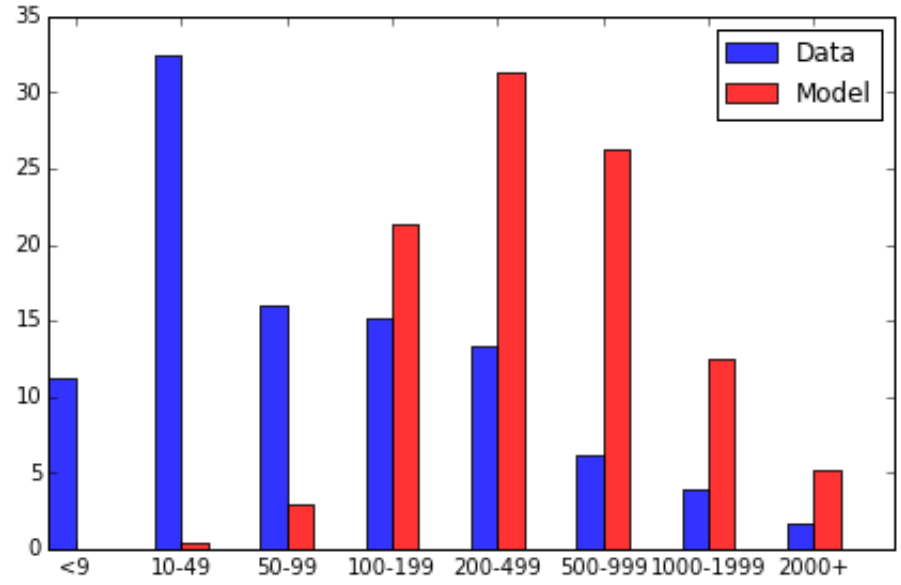
In [30]:
```
sCutsResults=np.zeros((7,1))
for i in range(0,7):
    sCutsResults[i]=newton(findsGivent,LandSizeCuts[i],fprime=None,arg
s=(LandSizeCuts[i]/10,AL,AK,PHI,RHO,GAMMA),tol=1e-8,maxiter=10000)
interOmeansTResult=np.zeros((8,1),dtype=float)
densityOTResult=np.zeros((8,1),dtype=float)
for i in range(0,8):
    if i==0:
        interOmeansTResult[i]=quad(intLand,0,sCutsResults[i],args=(MU,
SIGMA,AL,AK,PHI,RHO,GAMMA))[0]
        densityOTResult[i]=lognorm.cdf(sCutsResults[i],SIGMA,MU)
        interOmeansTResult[i]=interOmeansTResult[i]/densityOTResult[i]
    elif i==7:
        interOmeansTResult[i]=quad(intLand,sCutsResults[i-1],Inf,args=
(MU,SIGMA,AL,AK,PHI,RHO,GAMMA))[0]
        densityOTResult[i]=1-lognorm.cdf(sCutsResults[i-1],SIGMA,MU)
        interOmeansTResult[i]=interOmeansTResult[i]/densityOTResult[i]
    else:
        interOmeansTResult[i]=quad(intLand,sCutsResults[i-1],sCutsResu
lts[i],args=(MU,SIGMA,AL,AK,PHI,RHO,GAMMA))[0]              # We don't
 really need this...
        densityOTResult[i]=lognorm.cdf(sCutsResults[i],SIGMA,MU)-logno
rm.cdf(sCutsResults[i-1],SIGMA,MU)
        interOmeansTResult[i]=interOmeansTResult[i]/densityOTResult[i]
```

In [31]:
```
densityOTResult*100
```

Out[31]:
```
array([[  4.09080158e-04],
       [  3.58984383e-01],
       [  2.99494458e+00],
       [  2.13938033e+01],
       [  3.13375748e+01],
       [  2.62549121e+01],
       [  1.25098043e+01],
       [  5.14956747e+00]])
```

In [33]:

```
fig, ax = plt.subplots()

index = np.arange(8)
bar_width = 0.25

opacity = 0.8
error_config = {'ecolor': '0.3'}

rects1 = plt.bar(index, np.array(LandOctiles[0,:]), bar_width,
                 alpha=opacity,
                 color='b',
                 label='Data')

rects2 = plt.bar(index + bar_width, np.array(densityOTResult*100), bar
_width,
                 alpha=opacity,
                 color='r',
                 label='Model')
ax.set_xticks(index+bar_width)
ax.set_xticklabels( ('<9', '10-49', '50-99', '100-199', '200-499','500
-999','1000-1999','2000+') )


plt.tight_layout()
plt.legend()
plt.show()
```

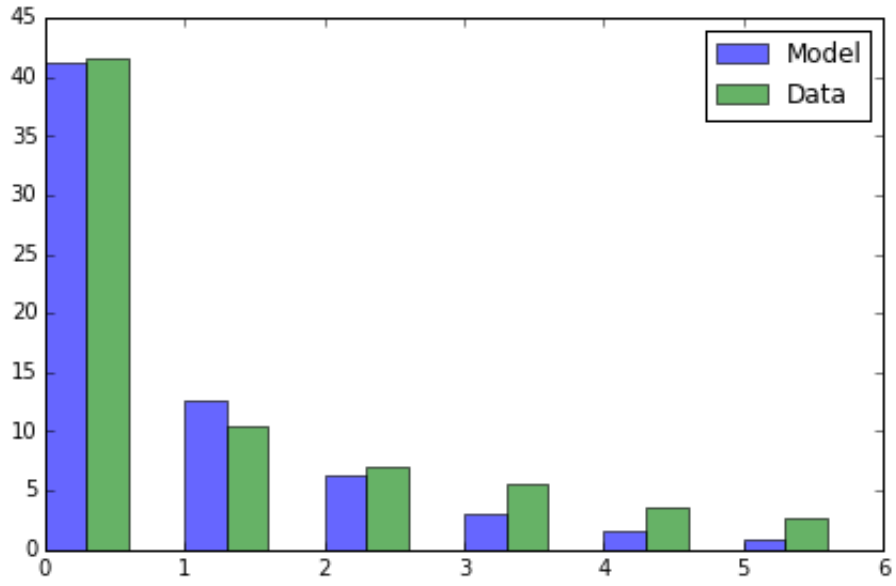


So, we can see that our model is not perfect in that it misses the data a bit. This suggests we may want to scale things differently in the minimization problem - give more weight, perhaps, to matching the density. What about the group-by-group mean factor demands?

In [34]:
```
sCutsLKResult=np.zeros((5,1))
for i in range(0,5):
    sCutsLKResult[i]=newton(findsGivent,LandSizeCutsLK[i],fprime=None,
args=(LandSizeCuts[i]/10,AL,AK,PHI,RHO,GAMMA),tol=1e-8,maxiter=10000)
interSmeansLResult=np.zeros((6,1),dtype=float)
interSmeansKResult=np.zeros((6,1),dtype=float)
densityOTResult=np.zeros((8,1),dtype=float)
densityOLKResult=np.zeros((6,1),dtype=float)
for i in range(0,6):
    if i==0:
        interSmeansLResult[i]=quad(intLabor,0,sCutsLKResult[i],args=(M
U,SIGMA,AL,AK,PHI,RHO,GAMMA))[0]
        interSmeansKResult[i]=quad(intCapital,0,sCutsLKResult[i],args=
(MU,SIGMA,AL,AK,PHI,RHO,GAMMA))[0]
        densityOLKResult[i]=lognorm.cdf(sCutsLKResult[i],SIGMA,MU)
        interSmeansLResult[i]=interSmeansLResult[i]/densityOLKResult[i
]
        interSmeansKResult[i]=interSmeansKResult[i]/densityOLKResult[i
]
    elif i==5:
        interSmeansLResult[i]=quad(intLabor,sCutsLKResult[i-1],Inf,arg
s=(MU,SIGMA,AL,AK,PHI,RHO,GAMMA))[0]
        interSmeansKResult[i]=quad(intCapital,sCutsLKResult[i-1],Inf,a
rgs=(MU,SIGMA,AL,AK,PHI,RHO,GAMMA))[0]
        densityOLKResult[i]=1-lognorm.cdf(sCutsLKResult[i-1],SIGMA,MU)
        interSmeansLResult[i]=interSmeansLResult[i]/densityOLKResult[i
]
        interSmeansKResult[i]=interSmeansKResult[i]/densityOLKResult[i
]
    else:
        interSmeansLResult[i]=quad(intLabor,sCutsLKResult[i-1],sCutsLK
Result[i],args=(MU,SIGMA,AL,AK,PHI,RHO,GAMMA))[0]
        interSmeansKResult[i]=quad(intCapital,sCutsLKResult[i-1],sCuts
LKResult[i],args=(MU,SIGMA,AL,AK,PHI,RHO,GAMMA))[0]
        densityOLKResult[i]=lognorm.cdf(sCutsLKResult[i],SIGMA,MU)-log
norm.cdf(sCutsLKResult[i-1],SIGMA,MU)
        interSmeansLResult[i]=interSmeansLResult[i]/densityOLKResult[i
]
        interSmeansKResult[i]=interSmeansKResult[i]/densityOLKResult[i
]
```

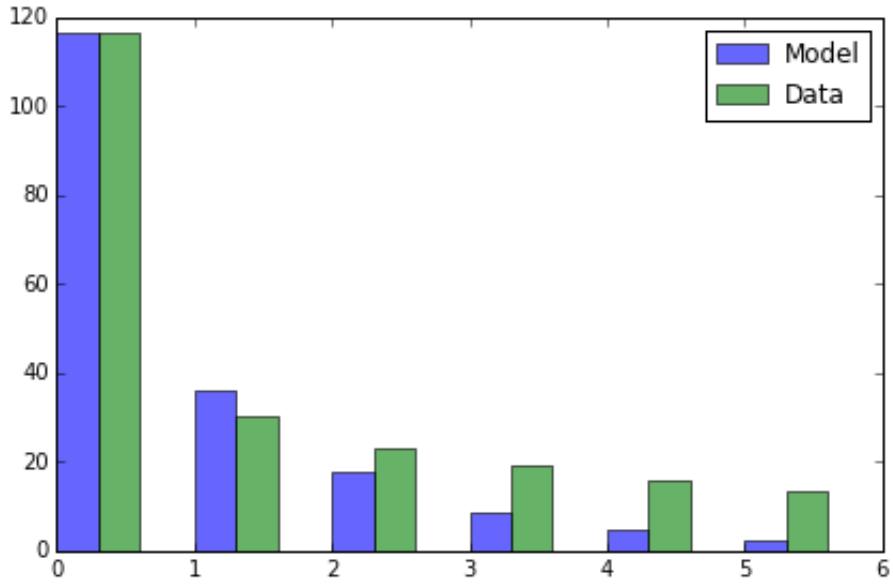
Let's graph these by bin and see how close they come to the data. First, labor demand by size grouping:

In [42]:
```
fig, ax = plt.subplots()

index = np.arange(6)
bar_width = 0.3

opacity = 0.6
error_config = {'ecolor': '0.3'}

rects1 = plt.bar(index, np.array(np.squeeze(interSmeansLResult)), bar_
width,
                 alpha=opacity,
                 color='b',
                 label='Model')

rects3 = plt.bar(index + bar_width, np.array(LaborTarget), bar_width,
                 alpha=opacity,
                 color='g',
                 label='Data')

plt.tight_layout()
plt.legend()
plt.show()
```



The labor fit looks pretty good (even though the bins don't seem to be lining up correctly (why is that?). What about Capital?

In [44]:
```
fig, ax = plt.subplots()

index = np.arange(6)
bar_width = 0.3

opacity = 0.6
error_config = {'ecolor': '0.3'}

rects1 = plt.bar(index, np.array(np.squeeze(interSmeansKResult)), bar_
width,
                 alpha=opacity,
                 color='b',
                 label='Model')

rects3 = plt.bar(index + bar_width, np.array(CapitalTarget), bar_width
,
                 alpha=opacity,
                 color='g',
                 label='Data')

plt.tight_layout()
plt.legend()
plt.show()
```



Close. While the model very roughly matches the data, the point is really that this sort of thing can be done. Moreover, the basic idea can be extended to solve for some of the things that we have taken for given, such as wages, capital rates, etc.

We have worked up an example here where we match binned data from the United States. These results can be extended to other parts of the world in several complementary ways. One idea, following Adamopolous and Restuccia, is to use the

skill distribution from the United States data (which we have essentially estimated) as "the frontier" in a model that can then be used to document departures from efficiency elsewhere in the world.

Another idea would be to fit a model like the above to specific regions or areas of a country with comparison of the distribution of "unobserved factors" across regions. If anomalies were observed, one could then explain them.

A final idea would be to bring additional information to bear on the problem and build sources of inefficiency directly into the model. Adamopolous and Restuccia do this for different parts of the world. Our gold standard would be including strategic behavior on the part of "large interests" in the model.

In [ ]:

A python class for economies with an equilibrium size distribution of Farms (Competitive market-power and/or tax-distorted)

Variations on a neo-classical model to explore how factor endowments, the initial distribution of property rights and skills in the population interact with production technologies to shape equilibrium patterns of agrarian production organization and the size distribution of farms.

To understand the main arguments, consider the simplest case of a single landlord (or a cartel of landlords) surrounded by a fringe of small landowning or landless agricultural households. If the lanlord owns a large fraction of the land endowment a standard partial-equilibrium analysis of non-price discriminating monopoly suggests the landlord would drive up the rental price of land by withholding land from the lease market. In a general equilibrium setting however there is another effect: by restricting other farmers' access to land landlords also lower the marginal product of labor on those farms. This increases the supply of labor to landlord estates at any given wage increasing landlords' potential income from monopsony rents. This can lead to equilibria where landlords increase the size of their production estates scale well above efficient scale in a competitive economy. A Latifundia-Minifundia type economy can emerge in which landlords operate large estates employing overly land-intensive production techniques while a large mass of farmers operate inefficiently and labor-intensive small parcels of land and sell labor to the landlord estate(s).

The model

The following is a close adaptation of earlier statement of this problem(Conning, 2004, 2010). We start with a simplified statement of the problem with just two types of households -- landlords and `peasants' and a very simple discrete distribution of farming skills. We later generalize.

Preliminaries

The economy has $\overline{T}$ units of cultivable land and $\overline{L}$ households with one unit of labor each. The economy-wide land to labor ratio is therefore $\bar{t} = \overline{T}/\overline{L}$.

Household are indexed $i = 1..\overline{L}$ and each household has a non-traded farming skill level s_i drawn from a known distribution Z. \ There is also an initial distribution of property rights over land. We'll make more flexible assumptions later but for now suffice to say that there is a group of "landlord" households (fraction λ of the total) who together own fraction θ of the land endowment. In some scenarios below they will collude to coordinate their factor market demands and supplies as if they were a single landlord `cartel'. \

As the $\lambda\overline{L}$ landlord households own $\theta\overline{T}$ units of land, peasant households own the remaining $(1-\theta)\overline{T}$ units. The average non-landlord household therefore owns $\frac{(1-\theta)\bar{t}}{(1-\lambda)}$ units and, for the moment, all peasant households have the same initial land endowment. Under these assumptions it can be shown that the land Gini coefficient is exactly $[\theta - \lambda]$.

A single tradable good such as corn is produced and consumed in the economy at a unity price fixed by trade with the world market. \ Households maximize utility from consumption subject to household income from farm production plus net factor sales.

All peasant households have access to the same production technology represented by a standard concave production function $\widehat{F}(T, L, s)$ assumed for now to be linearly homogenous in its three arguments: land T,labor L, and a third factor which we label s.This last factor is assumed to be a non-traded factor that captures farming skill or labor supervision ability.

In the illustrative simulations below we assume a Cobb-Douglas form

$$\widehat{F}(T, L, s) = s^{1-\gamma} \cdot \left[T^{\alpha} L^{1-\alpha}\right]^{\gamma}$$

Python implementation and Objects

The model in python builds off similar earlier efforts in Mathcad and MATLAB.

I employ object oriented programming ideas, first defining a "class" of Economy. An instance of an economy is an object with atributes including an associated endowment and technology as well as an initial distribution of property rights and non-traded skills. The economy class includes methods for finding a vector of market-clearing factor prices and associated equilibrium net factor demands and outputs.

I will later define a subclass PowerEconomy which inherits all the attributes of the Economy class but adds a few methods to compute market-power distorted equilibria.

In [1]:
```
%matplotlib inline
import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import minimize
```

```
from collections import namedtuple

np.set_printoptions(precision=4)

class Economy(object):
    """ Economy with an Equilibrium Farm Size Distribution
   At present the initial distribution of skills is uniformly distribu
ted.
   For example N = 5 and s = np.array([1, 1, 1, 1, 1.5]) has 5 farmer
groups.
   We take the landlord class to be last indexed group .
    """
    def __init__(self,N):  # constructor to set initial parameters.
                           # if None supplied use defaults
      self.N         = N # of quantiles (number of skill groups)
      self.GAMMA     = 0.8    # homogeneity factor
      self.ALPHA     = 0.4    # alpha (land) for production function
      self.LAMBDA    = 1.0/N    # landlord share of labor
      self.TBAR      = 100    # Total Land Endowment
      self.LBAR      = 100    # Total Labor Endowment
      self.H         = 0.0    # fixed cost of production

    def prodn(self,X,s):
        Y = s*((X[0]**self.ALPHA)*(X[1]**(1-self.ALPHA)))**self.GAMMA
        return Y

    def marginal_product(self,X,s):
        """ Production function technoogy """
        MPT = self.ALPHA*self.GAMMA*self.prodn(X,s)/X[0]
        MPL = (1-self.ALPHA)*self.GAMMA*self.prodn(X,s)/X[1]

        return np.append(MPT,MPL)

    def profits(self,X,s,w):
        """ Vector firm profits given factor prices and (T,L,s)"""
        return self.prodn(X,s) - np.dot(w,X) - self.H

    def demands(self,w,s):
        """Returns competitive demands for each skill group in a subec
onomy

        with factor supply (tbar, lbar) and vector s.
        """
        alpha, gamma = self.ALPHA, self.GAMMA
        land = ((w[1]/(gamma*s*(1-alpha))) *
                    (((1-alpha)/alpha)*(w[0]/w[1])) **
                     (1-gamma*(1-alpha)))**(1/(gamma-1))

        labor = ((w[0]/(gamma*s*alpha)) *
                       ((alpha/(1-alpha))*(w[1]/w[0])) **
                       (1-gamma*alpha))**(1/(gamma-1))
        # zero out demands if fixed cost implies negative profits
        X = np.array([land,labor])
```

```
        profitable = (self.profits(X,s,w)>0)
        return X*profitable


    def excessD(self,w,Xbar,s):
        """ Total excess land and labor demand given factor prices in
        subeconomy with Xbar supplies

        returns excess demand in each market
        """
        res = np.array(([np.sum(self.demands(w,s)[0])-Xbar[0], \
                 np.sum(self.demands(w,s)[1])-Xbar[1]]))
        #print(w,res)
        return res

    def competitive_eq(self,Xbar,s,analytic=True):  #numerically solve
d
        """ Solves for market clearing factor prices (analytically or
numerically)
        for a subeconomy with Xbar supplies

        If numerically: minimizes sum of squared excess demands
        Returns a named tuple with factor prices and demands
        c_eqn.w, c_eqn.D
        """

        if analytic:     # for specific CobbDouglas
            gamma=self.GAMMA
            #print('debug compet: ',Xbar)

            s_fringe, s_R = s[0:-1], s[-1]
            psi = np.sum((s_fringe/s_R)**(1/(1-gamma)))
            Lr = Xbar[1]/(1+psi)
            Tr = Xbar[0]/(1+psi)
            L_fringe = Lr*(s_fringe/s_R)**(1/(1-gamma))
            T_fringe = Tr*(s_fringe/s_R)**(1/(1-gamma))
            X = np.array([np.append(T_fringe,Tr),np.append(L_fringe,Lr
)])
            WR = self.marginal_product(X[:,-2],s[-2])

        else:  #Numerically.  works for any demands
            w0= np.array([0.5,0.5])
            f = lambda w: np.sum(self.excessD(w,Xbar,s)**2)
            res = minimize(f,w0,method='Nelder-Mead')
            WR=res.x

        X=self.demands(WR,s)
        result = namedtuple('result',['w','D'])
        res = result(w=WR,D=X)
        return res


    def cartel_income(self,Xr,theta):
```

```
        """ Cartel group's income from profits and factor income

        when cartel uses (tr,lr) fringe has (TBAR-tr,LBAR-lr)  """
        #at present cartel is always last index farm
        s_fringe, s_R = s[0:-1], s[-1]  #landlord is top index farmer

        TB_fringe = max(self.TBAR - Xr[0],0)
        LB_fringe = max(self.LBAR - Xr[1],0)
        fringe  = self.competitive_eq([TB_fringe,LB_fringe],s_fringe)
        y= self.prodn(Xr,s_R) - \
            np.dot(fringe.w,[Xr[0]-self.TBAR*theta,Xr[1]-self.LAMBDA*s
elf.LBAR])
        # print("cartel:  Tr={0:8.3f}, Lr={1:8.3f}, y={2:8.3f}".format
(Xr[0],Xr[1],y))
        return y

    def cartel_eq(self,theta,guess=[1,1]):
        """ Cartel chooses own factor use (and by extension how much t
o
        withold from the fring to max profits plus net factor sales)
        """
        f = lambda X: -self.cartel_income(X,theta)
        res = minimize(f,guess,method='Nelder-Mead')
        XR = res.x
       # print('XR:',XR)
        fringe = self.competitive_eq([self.TBAR,self.LBAR]-XR,s[0:-1])
        X_fringe = fringe.D
        WR =fringe.w

        result = namedtuple('result',['Xr','w','Xf'])
        cartel_res = result(Xr=XR,w=WR,Xf=X_fringe)
        return cartel_res
```

Latifundia Economics (factor market power distorted equilibria)

The following contrived example helps to starkly highlight the mechanisms behind the factor-market power distorted equilibrium.

For the moment we simply assume that there are N=100 farmers each with a skill level normalized to 1.

In [2]:
```
N=5
s = np.ones(N)
```

Now create an economy and change a few parameters from their default. The γ parameter which measures the degree of homogeneity in production is purposefully set very high.. We are very close to assuming constant returns to scale (but setting it just below 1 is needed to make sure the size-distribution remains determinate).

In [3]:
```
E = Economy(N)
E.ALPHA = 0.4
E.GAMMA =0.98
```

The Economy has resource endowment:

In [4]:
```
E.TBAR,E.LBAR
```

Out[4]:
```
(100, 100)
```

So as expected the efficient (competitive) resource allocation has every farmer operating a farm of equal unit size.

In [5]:
```
Xc = E.competitive_eq([E.TBAR,E.LBAR],s)
```

Thus far we've said nothing of the ownership of land or labor. Let's assume every household has one unit of labor but that the 'landlord' class (which WLOG we index to be the last skill group s[-1]) owns fraction θ of the land. Assuming a uniform distribution of households across skills every skill group has Lbar/N households, and so there are that many landlords who act as a single cartel.

The following code is useful for printing and plotting out equilibria as a function of initial landlords' land ownership share.

In [6]:
```
def scenarios(ECO,numS=5,prnt=True,detail=True):
        """Creates numS land ownership (theta) scenarios
        and returns competitive and market-power distorted equilibria

        Prints results if flags are on.

        Args:
          numS -- number of values of theta
          prnt -- print table if True
        Returns:
          [Xc,Xr,wc,wr]  where
            Xrc -- Efficient/Competitive landlord factor use
            Xr -- numS x 2 matrix, Xr[theta] = Landlords' distorted us
e
            wc -- competitive factor prices
            wr -- wr[theta] distorted competitive factor prices

        """
        print("Running {0} scenarios...".format(numS))

        # competitive eqn when landlord is just part of the competitiv
e fringe
        comp = ECO.competitive_eq([ECO.TBAR,ECO.LBAR],s)
```

```
        wc = comp.w
        Xc = comp.D
        Xrc = Xc[:,-1]   # landlord's factor use

        # Market-power distorted equilibria for different theta
        guess = Xrc
        theta = [1.0*i/(numS) for i in range(numS+1)]

        if prnt:
            print("\nAssumed Parameters")
            print("==================")
            params = E.__dict__
            for p in params:
                print("{0:<7} : {1}".format(p,params[p]))
            print('\nEfficient [ Trc, Lrc]      [rc,wc]        w/r   '
),
            if detail:
                print('F( )  r*Tr]        w/r'),
            print("")
            print("="*80)
            print("          [{0:6.2f},{1:6.2f}] ".format(Xrc[0],Xrc[1
])),
            print("[{0:4.2f},{1:4.2f}]".format(wc[0],wc[1])),
            print("   {0:4.2f} ".format(wc[1]/wc[0])),
            if detail:
                print('F( )  r*Tr]        w/r'),
                print("| {0:5.2f} ".format(ECO.prodn(Xrc,s[-1]))),
                print(" {0:5.2f} ".format(Xrc[0]*wc[0])),

            print("\n\nTheta  [ Tr, Lr ]      [rM,wM]        w/r  |"),
            print('F()   [T_hire]  [T_sale] [L_hire]')

            print("="*80)

        Xr = np.zeros(shape=(numS+1,2))  # Xr - lord factor use for ea
ch theta
        wr = np.zeros(shape=(numS+1,2))
        for i in range(numS+1):
            cartelEQ = E.cartel_eq(theta[i],guess)
            Xr[i] = cartelEQ.Xr
            wr[i] = cartelEQ.w
            guess = Xr[i]
            if prnt:
                print(" {0:3.2f}".format(theta[i])),
                print(" [{0:6.2f},{1:6.2f}]".format(Xr[i,0],Xr[i,1])),
                print("[{0:5.2g},{1:5.2f}] {2:5.2f}" \
                .format(wr[i,0],wr[i,1],wr[i,1]/wr[i,0])),
                if detail:
                    print("| {0:5.2f} ".format(ECO.prodn(Xr[i],s[-1]))
),
                    print(" {0:5.2f} ".format(Xr[i,0]*wr[i,0])),
                    print(" {0:5.2f} ".format(theta[i]*ECO.TBAR*wr[i,0
])),
```

```
                    print(" {0:6.2f} ".format(Xr[i,1]*wr[i,1])),
                print("")
        if prnt:
            print("=======================================+++====\n")

        return (Xrc,Xr,wc,wr)

def scene_plot(ECO,Xrc,Xr):
    numS = len(Xr)-1
    theta = [1.0*i/(numS) for i in range(numS+1)]
    Tr,Lr = Xr[:,0],Xr[:,1]
    Tr_net = Tr-np.array(theta)*ECO.TBAR
    Lr_net = Lr -ECO.LAMBDA*ECO.LBAR
    print(Tr_net,Lr_net)
    Trc_net = Xrc[0]*np.ones(numS+1)-np.array(theta)*E.TBAR
    Lrc_net = Xrc[1]*np.ones(numS+1)-ECO.LAMBDA*ECO.LBAR
    plt.grid()
    plt.plot(theta,Tr_net,'-ro')
    plt.plot(theta,Trc_net)
    plt.plot(theta,Lr_net,'-bx')
    plt.plot(theta,Lrc_net)
    plt.grid()
    plt.ylim(-100,ECO.TBAR)
    #plt.xlabel(r'$\gamma =$')
    plt.title('Landlord factor use for '+r'$\gamma =$ {0}'.format(ECO.
GAMMA))
    plt.xlabel(r'$\theta$ -- Landlord land ownership share')
    plt.show()
    return
```

We pass our economy instance to the scenarios function and it solves for competitive and market-power equilibria (at different levels of theta) and prints out the results.

In [7]:
```
(Xrc,Xr,wc,wr) = scenarios(E,10,detail=True)
```

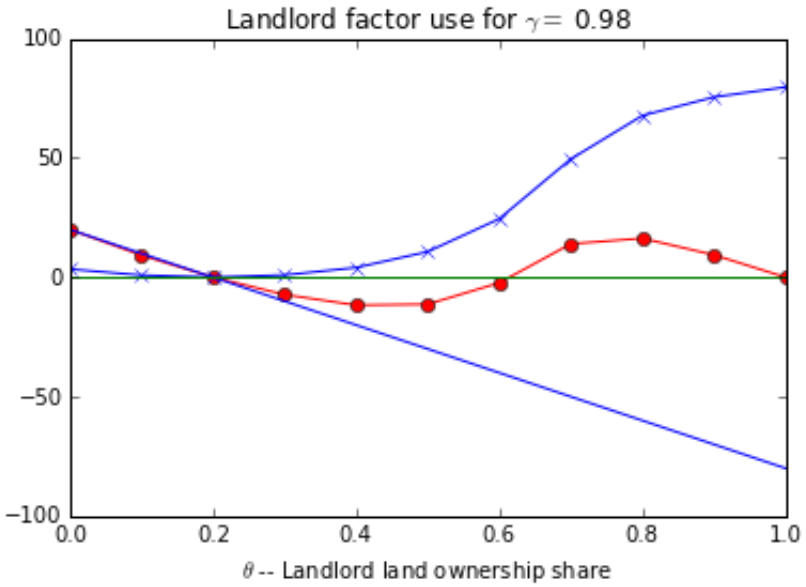
```
Running 10 scenarios...

Assumed Parameters
==================
TBAR    : 100
H       : 0.0
LBAR    : 100
ALPHA   : 0.4
N       : 5
GAMMA   : 0.98
LAMBDA  : 0.2

Efficient [ Trc, Lrc]      [rc,wc]        w/r    F( )  r*Tr]        w/
r
=================================================================
==========

          [ 20.00  20.00]  [0.37 0.55]     1.50  F( )  r*Tr]        w/r
```

```
          [ 20.00, 20.00]  [0.37,0.55]    1.50  F( )  r*Tr]         w/r
 | 18.84    7.38

Theta  [ Tr, Lr ]      [rM,wM]        w/r  | F()   [T_hire]  [T_sale]
[L_hire]
=========================================================================
==========
 0.00  [ 19.99, 23.40] [ 0.36, 0.56]  1.57 | 20.65    7.19    0.00
13.19
 0.10  [ 19.21, 20.85] [ 0.36, 0.56]  1.53 | 19.00    7.01    3.65
11.64
 0.20  [ 20.00, 20.00] [ 0.37, 0.55]  1.50 | 18.84    7.38    7.38
11.08
 0.30  [ 22.73, 20.91] [ 0.37, 0.55]  1.47 | 20.33    8.51   11.24
11.48
 0.40  [ 28.31, 24.03] [ 0.38, 0.54]  1.42 | 24.05   10.84   15.31
13.02
 0.50  [ 38.68, 30.68] [  0.4, 0.53]  1.33 | 31.37   15.43   19.95
16.24
 0.60  [ 57.64, 44.45] [ 0.44, 0.50]  1.14 | 45.62   25.28   26.31
22.29
 0.70  [ 83.97, 69.48] [ 0.56, 0.44]  0.79 | 68.75   46.75   38.97
30.48
 0.80  [ 96.28, 87.70] [ 0.79, 0.36]  0.45 | 83.18   76.39   63.47
31.54
 0.90

-c:31: RuntimeWarning: invalid value encountered in double_scalars
-c:48: RuntimeWarning: invalid value encountered in double_scalars
-c:48: RuntimeWarning: invalid value encountered in power
-c:52: RuntimeWarning: invalid value encountered in double_scalars
-c:52: RuntimeWarning: invalid value encountered in power
-c:26: RuntimeWarning: invalid value encountered in power
-c:55: RuntimeWarning: invalid value encountered in greater
-c:32: RuntimeWarning: invalid value encountered in double_scalars

  [ 99.37, 95.45] [  1.3, 0.27]  0.21 | 88.51   129.07   116.90    25.
81
 1.00  [100.00, 99.62] [4e+07, 0.00]  0.00 | 91.00   3952518284.45   3
952518284.45     0.00
=======================================++====
```

Let's plot the results.. THis is the classic diagram from Latifundia Economics..

In [8]:
```
scene_plot(E,Xrc,Xr)
```

```
(array([  1.9985e+01,   9.2088e+00,  -3.3124e-05,  -7.2672e+00,
        -1.1689e+01,  -1.1323e+01,  -2.3595e+00,   1.3969e+01,
         1.6284e+01,   9.3678e+00,  -2.8422e-14]), array([  3.4010e+00
,   8.4540e-01,  -2.8245e-05,   9.1067e-01,
         4.0302e+00,   1.0681e+01,   2.4446e+01,   4.9480e+01,
         6.7702e+01,   7.5446e+01,   7.9616e+01]))
```



Now let's make the landlord a bit more 'skilled'. This lowers the cost of being big. But it also makes him bigger at lower theta and makes the size/Feenstra effect kick in (i.e. landlord farm may be relatively *small* at low theta.

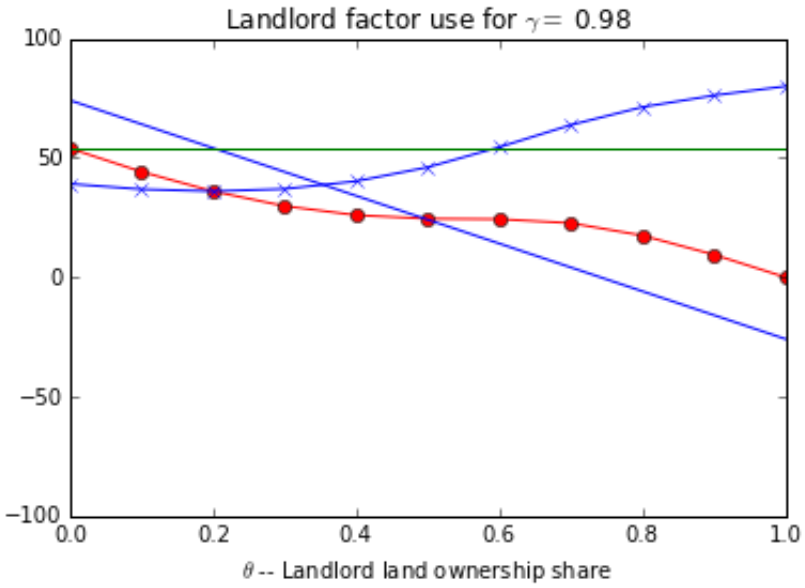
In [9]:
```
s[-1]=1.05
```

In [10]:
```
(Xrc,Xr,wc,wr) = scenarios(E,10,detail=True)
```

```
Running 10 scenarios...

Assumed Parameters
==================
TBAR    : 100
H       : 0.0
LBAR    : 100
ALPHA   : 0.4
N       : 5
GAMMA   : 0.98
LAMBDA  : 0.2

Efficient [ Trc, Lrc]      [rc,wc]        w/r    F( )  r*Tr]        w/
r
=================================================================
==========
          [ 74.14, 74.14]  [0.38,0.57]    1.50  F( )  r*Tr]        w/r
 | 71.42   28.00

Theta  [ Tr, Lr ]      [rM,wM]        w/r  | F()   [T_hire]  [T_sale]
[L_hire]
=================================================================
==========
 0.00  [ 53.99, 59.11] [ 0.35, 0.59]  1.69 | 55.21   18.80    0.00
34.75
 0.10  [ 54.19, 56.85] [ 0.36, 0.57]  1.59 | 54.04   19.53    3.60
32.63
 0.20  [ 56.01, 56.01] [ 0.37, 0.56]  1.50 | 54.26   20.93    7.47
31.39
 0.30  [ 59.83, 56.99] [ 0.39, 0.55]  1.40 | 56.25   23.31   11.69
31.11
 0.40  [ 66.00, 60.21] [ 0.41, 0.53]  1.28 | 60.38   27.19   16.48
31.79
 0.50  [ 74.52, 66.07] [ 0.45, 0.50]  1.13 | 66.88   33.32   22.35
33.27
 0.60  [ 84.33, 74.48] [ 0.51, 0.47]  0.92 | 75.32   42.85   30.49
34.85
 0.70  [ 92.68, 83.78] [ 0.62, 0.42]  0.68 | 83.77   57.32   43.29
35.06
 0.80  [ 97.49, 91.27] [ 0.82, 0.36]  0.43 | 89.85   80.30   65.89
32.41
 0.90  [ 99.47, 96.15] [  1.3, 0.27]  0.21 | 93.38   130.62   118.18
  25.96
 1.00  [100.00, 99.98] [1.1e+07, 0.00]  0.00 | 95.75   1123661480.15
 1123661480.15     0.00
=========================================+++====
```

In [11]:
```
scene_plot(E,Xrc,Xr)
```

```
(array([  5.3990e+01,   4.4195e+01,   3.6006e+01,   2.9830e+01,
         2.5999e+01,   2.4520e+01,   2.4331e+01,   2.2684e+01,
         1.7490e+01,   9.4722e+00,  -1.4211e-14]), array([ 39.1135,  3
6.8535,  36.006 ,  36.9885,  40.2128,  46.0661,
        54.4779,  63.7807,  71.2656,  76.1504,  79.9779]))
```



Adamopoulos and Restuccia AER 2014 (port from MATLAB)

Our first Ipython effort. Matt Baker did most of the heavy lifting of porting the A&R model from MATLAB. Their MATLAB code was not very readable (and at times downright cryptic) nor very efficient so we've cleaned things up a bit along the way.

In [3]:
```
%pylab inline
from __future__ import division
```

```
Populating the interactive namespace from numpy and matplotlib
```

The model starts with a description as to how the works.

But first output in the non-agricultural sector is modeled this way:

Production is Cobb-Douglas: $Y_n = AK_n^{\alpha}N_n^{1-\alpha}$

This implies that we have simple expressions for the economy-wide wage w and rental rate on capital r:

$r = MPK = \alpha AK_n^{\alpha}N_n^{1-\alpha}$

which becomes:

$r = \alpha A\left(\frac{K_n}{N_n}\right)^{\alpha-1}$

while we also have:

$w = MPL = (1-\alpha)A\left(\frac{K_n}{N_n}\right)^{1-\alpha}$

The next part of the model is the agricultural sector. Here, we have heterogeneity in farm managers characterized by a "skill" variable s. The farm production function is:

$y_a = A\kappa[\theta k^{\rho} + (1-\theta)(sl)^{\rho}]^{\frac{\gamma}{\rho}}$

As Jonathan notes, this is a little different than Lucas's (1978) span of control, as s here is augmenting l - land - not overall production. That is, s is not outside the brackets.

Before getting into solution, we can start the process of coding things up in Julia. Some simple aspects of the calibration involve normalization of κ and A at unity. We also have values of α and γ set from outside sources:

In [4]:
```
KAPPA_US=1
A_US=1
ALPHA=0.33
GAMMA=0.54
```

Some additional information that is required are distribution of the size of farms. This information comes in two pieces. First, there is the fraction of each farm in each size class. Then, there is the distribution of land by size class. These two pieces of information are as follows (duly noting that I'm not exactly sure how these interact in the model at this point):

In [5]:
```
farm_pdf_data=[.1056, .2813, .0698, .0871, .0794, .0633, .0397, .0310,
 .0964, .0679, .0420, .0365]
```

As alluded to above, the actual size of the bins are as follows:

In [6]:
```
land_pdf_data=[.0012, .0173, .0097, .0171, .0220, .0238, .0187, .0176,
 .0823, .01129, .01384, .05389 ]
```

Some additional calibrated variables include: The average farm size (in the United States), The share of labor in agriculture in the United States, The United States capital labor ratio, The ratio of capital land ratio between smallest and largest farms, The ratio of agricultural to non-agricultural labor productivity, The long-run share of labor in agriculture, And the agricultural land income share.

Respectively, these variables are:

In [7]:
```
AFS_US=169.249
Na_US=.025
KY_US=2.5
kl_min_max_data=84.85
paYNan_ratio=1.0/2.0
PHI=0.010
L_inc_sh=0.18
```

The calibrated variables are then used to determine many of the quantities that are actually used in modeling. For one, we have the mysterious quantity:

In [8]:
```
rKa_qL_US=GAMMA/L_inc_sh-1
```

Jonathan has pointed out that this is funny little equation results from Euler's Theorem: total factor payments add up to γ of output, with profits being what is left after this. Accordingly:

$$rK + qL = \gamma F_K K + \gamma F_L N = \gamma Y_a$$

This implies:

$$\frac{rK}{qL} + 1 = \frac{\gamma Y_a}{qL}.$$

We then have the above rKa_ql_US as the term $\frac{rK}{qL}$ in the previous equation:

$$\frac{rK}{qL} = \frac{\gamma Y_a}{qL} - 1.$$

A next equation that is a little fishy is the following "derived calibration":

In [9]:
```
YN_US=(A_US**(1/(1-ALPHA)))*KY_US**(ALPHA/(1-ALPHA))
```

In terms of nice formatting, I'm thinking this is worker productivity. That is, we have:

$$\frac{Y_n}{N_n} = A^{\frac{1}{1-\alpha}} \left(\frac{K_n}{Y_n}\right)^{\frac{\alpha}{1-\alpha}}$$

The last expression is aggregate labor productivity, but it is hard to see where an expression like this could come from. But first, let's think about the goal, which is to write labor productivity as a function of the capital-output ratio. If we started with a function like:

$$Y_n = AK_n^{\alpha} N_n^{1-\alpha}$$

Now "solve" this for N_n and write this as:

$$Y_n^{\frac{1}{1-\alpha}} = A^{\frac{1}{1-\alpha}} N_n K_n^{\frac{\alpha}{1-\alpha}}$$

Now, we note that we can write this as follows:

$$Y_n Y_n^{\frac{\alpha}{1-\alpha}} = A^{\frac{1}{1-\alpha}} N_n K_n^{\frac{\alpha}{1-\alpha}}$$

Now divide through by $Y_n^{\frac{\alpha}{1-\alpha}} N_n$ to get the desired result.

The few quantities of interest are calibrated as follows:

In [10]:
```
K_US=KY_US*YN_US
LN_US=AFS_US*Na_US
one_minus_XI=(1-GAMMA)/(1-ALPHA)*paYNan_ratio
XI=1-one_minus_XI
```

In [11]:
```
one_minus_XI
```

Out[11]:
```
0.34328358208955223
```

What is this variable ξ? It is called the "value productivity ratio" in the paper. The operative assumption is that working in nonagriculture is subject to a tax. So non-agricultural worker $w(1-\xi)$.

It is a little hard for me to see why this is needed or where it is coming from at the moment. Anyways, we start out with a grid of guesses for μ and σ, and also provide some other details of the search for the mean and variance:

In [12]:
```
import numpy as np
mean=np.linspace(-1.836,-1.8,20)
var=np.linspace(4.6553,4.75,20)
DIFF=1000
C=np.zeros((12,1))
MU_min=0.0
SIGMA_min=0.0

MU=mean[1]
SIGMA=var[1]
```

Actually, the above entries should be looped over, but aren't in the code, which is presumably operating from the optimum values of μ and σ. Now, we make a grid of points and approximate a log-normal. First, we need a couple of functions that create a discretized log-normal distribution. While there are probably better ways to do this than following exactly what R/A did,let's in fact just follow along. First, let's get the package we need:

In [13]:
```
from scipy.integrate import quad
import math
```

We now build two functions. The first gives a log-normal pdf, while the second gives an approximate value for the cdf at the given point. Not how it might be beneficial to bring in stuff like "log" right off the bat, rather than having to use the "np" handle every time. Also note how ^ (squared) is instead **.

In [14]:
```
def lognorm_pdf(x,m,v):
    s=np.sqrt(v)
    out1=np.divide(1.,np.multiply(x,s*np.sqrt(2.*math.pi)))
    out2=np.exp(np.divide((-(np.log(x)-m)**2),(2*v)))
    return out1*out2
```

The above might be viewed as a more efficient coding of the lognormal pdf than R/A use. We also need a function that gives an approximation to the integrated lognormal:

In [15]:
```
def lognorm_discrete(x1,x2,m,v):
    out=quad(lognorm_pdf,x1,x2,(m,v))
    return out
```

Now, we make a little loop that approximates the log normal we programmed above

In [16]:
```
s_n=6000
s_grid=np.logspace(-5.5,2.,s_n)
s_prob=np.zeros((s_n,1))
VAR_s=SIGMA**2
```

We do have to endulge a few shenanigans here, mainly because s_grid above is not really a matrix as we would like it to be, but really just an array filled full of a list of numbers. Accordingly, let's convert it into the same form as the variables we are deriving from it:

In [17]:
```
s_grid=np.asmatrix(s_grid)
s_grid=s_grid.T
np.shape(s_grid)
```

Out[17]:
```
(6000L, 1L)
```

In [18]:
```
s_grid[0]
```

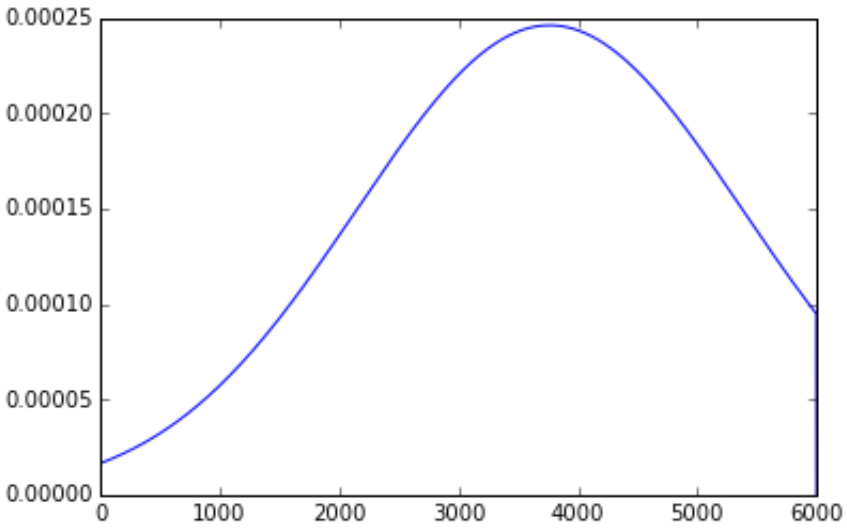
Out[18]:
```
matrix([[  3.16227766e-06]])
```

Now, we loop over the above and "fill in" the probability grid so that we have a whole probability distribution:

In [19]:
```
for i in range(0,s_n-1):
    if i==0:
        d=(s_grid[1]-s_grid[0])/2
        s_prob[i]=lognorm_discrete(s_grid[i]-d,s_grid[i]+d,MU,VAR_s)[0
]
    else:
        d=(s_grid[i]-s_grid[i-1])/2
        s_prob[i]=lognorm_discrete(s_grid[i]-d,s_grid[i]+d,MU,VAR_s)[0
]
```

Let's see what this looks like once we have approximated it. First, we import the plot library, and then just plot the numbers:

In [20]:
```
import matplotlib.pyplot as plt
plt.plot(s_prob)
plt.show()
```



I can't see where I've made a mistake, so perhaps that is correct, even though it looks a little goofy. Continuing on, we now get the sum of all the probabilities and see if they sum to unity:

In [21]:
```
np.sum(s_prob)
```

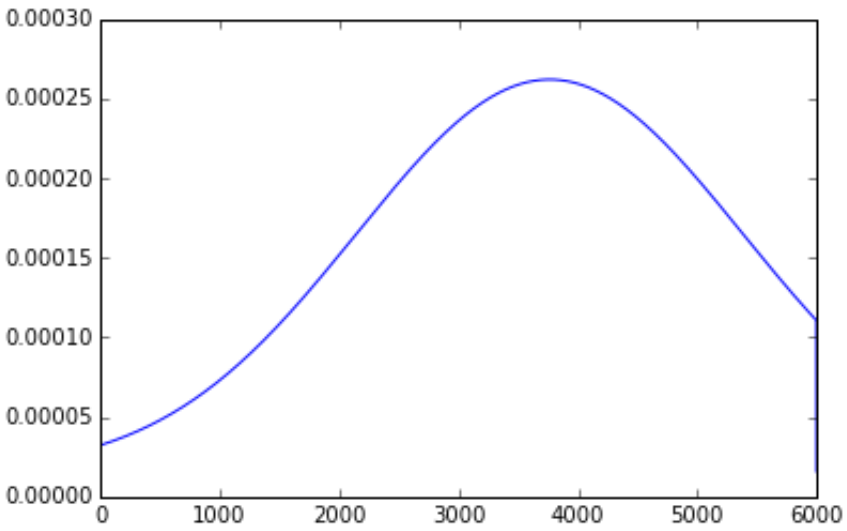
Out[21]:
```
0.90506654100125528
```

Thus, we have to renormalize, even though this is starting to seem a little sketchy...

JC: XX in the MATLAB code is 0.90539 so ever so slightly off..

In [22]:
```
XX=sum(s_prob)
if XX<1:
    s_prob=s_prob+(1-XX)/s_n
```

In [23]:
```
plt.plot(s_prob)
plt.show()
```



Because I find it perplexing that there is such a large truncation region of the graph, let's see if we can replicate the results with actual functions.

In [24]:
```
import scipy.stats as sps
```

Now, in Python, the log-normal is denoted a bit differently than R/A denote it. The python function uses parameters sc, sh, and lo together as follows:

$$f(x|sh,lo,sc) = \frac{1}{(x-lo)sh\sqrt{2\pi}} e^{\frac{-(\ln(x-lo)-ln(sc))^2}{2sh^2}}$$

So, if the Python function is to correspond with R/A's function, we need $lo = 0$. Moreover, since $\mu = ln(sc)$, we need $sc = e^{\mu}$. Finally, R/A parameterize the variance, where $v = sh^2$. Therefore, we need $sh = \sqrt{v}$. To check this out:

In [25]:
```
print sps.lognorm.pdf(2,np.sqrt(2),0,np.exp(1))
print lognorm_pdf(2,1,2)
print sps.lognorm.cdf(2,np.sqrt(2),0,np.exp(1))
```

```
0.13776596115
0.13776596115
0.414112858205
```

We can also use the lognorm CDF function to perhaps do a bit better than R/A, or at least check their work. We want the mass between the "slices," so:

In [26]:
```
s_prob2=np.zeros((s_n,1))
for i in range(0,s_n-1):
    if i==0:
        d=(s_grid[1]-s_grid[0])/2
        s_prob2[i]=sps.lognorm.cdf(s_grid[i]+d,np.sqrt(VAR_s),0,np.exp
(MU))-sps.lognorm.cdf(s_grid[i]-d,np.sqrt(VAR_s),0,np.exp(MU))
    else:
        d=(s_grid[i]-s_grid[i-1])/2
        s_prob2[i]=sps.lognorm.cdf(s_grid[i]+d,np.sqrt(VAR_s),0,np.exp
(MU))-sps.lognorm.cdf(s_grid[i]-d,np.sqrt(VAR_s),0,np.exp(MU))
```

While we are at it... let's try something even simpler

In [27]:
```
np.exp(MU)
```

Out[27]:
```
0.1597563777573214
```

In [28]:
```
s_prob3=np.zeros((s_n,1))
d = s_grid[1:]-s_grid[:-1]
C_top = sps.lognorm.cdf(s_grid[1:],np.sqrt(VAR_s),0,np.exp(MU))
C_low = sps.lognorm.cdf(s_grid[:-1],np.sqrt(VAR_s),0,np.exp(MU))
s_prob3 = C_top - C_low
s_prob3 =  np.append(s_prob2[0],s_prob3)
```

In [29]:
```
len(s_prob3)
s_prob3
```

Out[29]:
```
array([  1.65805164e-05,   1.65685468e-05,   1.65923419e-05, ...,
         9.50706677e-05,   9.49896307e-05,   9.49086266e-05])
```

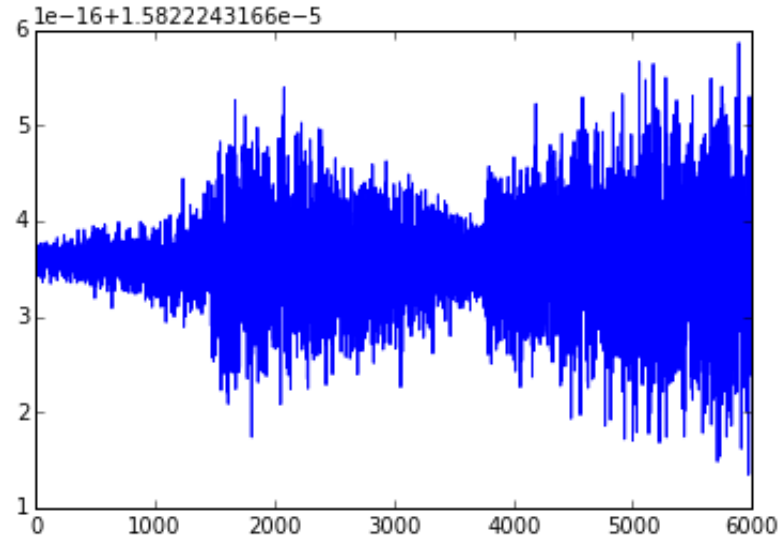
In [30]:
```
s_prob2[0],s_prob2[1],s_prob2[2],s_prob3[0],s_prob3[1]
```

Out[30]:
```
(array([  1.65805164e-05]),
 array([  1.65566015e-05]),
 array([  1.65803762e-05]),
 1.6580516426216441e-05,
 1.6568546826193492e-05)
```

In [59]:
```
type(s_prob3)
```

Out[59]:
```
numpy.ndarray
```

In [31]:
```
difference=abs(s_prob-s_prob2)
plt.plot(difference)
plt.show()
```



In [32]:
```
difference=abs(s_prob2-s_prob3)
#plt.plot(difference)
#plt.show()
difference.max()
```

Out[32]:
```
0.00024643122452328026
```

So they are very close, a couple of thousandths off. It is also encouraging to know we got the programming correct. Let's now try to code in the calibration_eval function that R/A use. We also need some guesses for other stuff, such as ρ

In [33]:
```
RHO=.24
```

Because one has to be very careful with certain types of objects (particularly matrices), I'm going to break up the calibration_eval function a little bit more than R/A do.

In [34]:
```
def calibration_eval(x,rKa_qL_US,Na_US):

    KNn_US=x[0]                                 # non-ag capital-labor
    THETA=x[1]                                  # parameter in Ag producti
on
    pa_US=x[2]                                  # relative price of agricu
ltural goods

    w_US=(1-ALPHA)*A_US*KNn_US**ALPHA       # w = MPLNa
    r_US=ALPHA*A_US*KNn_US**(ALPHA-1)
    Ka_US=K_US-(KNn_US*(1-Na_US))
    KNa_US=Ka_US/Na_US
    YNn_US=A_US*(KNn_US**ALPHA)
                #not used?
    q_r_US=(rKa_qL_US**(-1))*Ka_US/LN_US
    q_US=q_r_US*r_US
    aBAR=(1/(pa_US*(1-PHI)))*((w_US*(1-XI)*Na_US/((1-GAMMA)))-PHI*(w_U
S*(1-XI)+q_US*LN_US+r_US*K_US))  #not used?
    psi_s_vec1=THETA*((THETA*q_r_US/(1-THETA))**(RHO/(1-RHO)))*np.ones
((s_n,1))
    psi_s_vec2=(1-THETA)*np.power(s_grid,(RHO/(1-RHO)))
    psi_s_vec=psi_s_vec1+psi_s_vec2
    l_vec1=((pa_US*GAMMA*(1-THETA)*KAPPA_US*A_US/q_US)**(1/(1-GAMMA)))
    l_vec2=(np.power(psi_s_vec,((GAMMA-RHO)/(RHO*(1-GAMMA)))))
    l_vec3=(np.power(s_grid,(RHO/(1-RHO))))
    l_vec=l_vec1*np.multiply(l_vec2,l_vec3)
    k_l_vec=((THETA*q_r_US/(1-THETA))**(1/(1-RHO)))*((np.power(s_grid,
(-RHO/(1-RHO)))))
    k_vec = np.multiply(k_l_vec , l_vec)
    ya_vec=KAPPA_US*A_US*np.power((THETA*np.power(k_vec,RHO)+(1-THETA)
*np.power(np.multiply(s_grid,l_vec),RHO)),GAMMA/RHO)
    PI_vec=(1-GAMMA)*pa_US*ya_vec
    f1=(1-XI)*w_US-np.sum(np.multiply(s_prob,PI_vec))
    f2=KNa_US - np.sum(np.multiply(s_prob,k_vec))
    f3=LN_US - Na_US*(np.sum(np.multiply(s_prob,l_vec)))
    f=f1,f2,f3
    g=np.dot(f,f)
    return g
```

JC: As of 1/21 it is reproducing the matlab evaluation call exactly.

In [35]:
```
guess = array([3.2,0.9,0.4])
calibration_eval(guess,rKa_qL_US,Na_US)
```

Out[35]:
```
949.4260053043738
```

JC: MATLAB calibration_eval gives (when RHO=0.24):

```
calibration_eval(guess,rKa_qL_US,Na_US)
```

```
ans =

      0.26683       39.719       4.1983
```

In [36]:
```
import scipy.optimize as so
```

In [37]:
```
x0=np.array([3.2,.84,.27])
a=rKa_qL_US,Na_US
result=so.minimize(calibration_eval,x0,args=a,method='Nelder-Mead')
```

```
-c:15: RuntimeWarning: invalid value encountered in double_scalars
-c:18: RuntimeWarning: invalid value encountered in double_scalars
-c:19: RuntimeWarning: invalid value encountered in power
-c:22: RuntimeWarning: invalid value encountered in double_scalars
-c:24: RuntimeWarning: invalid value encountered in power
```

In [38]:
```
result
```

Out[38]:
```
  status: 0
    nfev: 210
 success: True
     fun: 5.8793619412223842e-08
       x: array([ 3.97104018,  0.8886881 ,  0.31545382])
 message: 'Optimization terminated successfully.'
     nit: 118
```

It seems then that we are able to reproduce results as obtained by A/R. The next thing we might like to do is see how the rest of the project goes. What we are now going to do is try to get a function that does what the rest of the calibration exercise does, but have this actually be a function, and not just an external loop. So, in essence, the function should work as follows:

1. Arguments taken in by the function should be ρ, μ, and σ.
2. The first order of business is then simulating a μ, σ log-normal distribution as done in the paper.
3. I don't see any reason why rKa_qL_US and Na_US should be afforded special status as arguments. Can't they be globals as well?
4. Given a simulated distribution, and the fixed values of rKa_qL_US and Na_US, the next thing to be done is to solve for the x-vector given these values (using a nested optimization routine, as above)
5. Given the results, we then run the rest of A/R's code and "match moments."

Before getting started, we will first define bins ahead of time, exploiting Python's nice way of working with scope. I am also going to try and improve the code a bit where I think it needs it. Here are the bins:

In [39]:
```
bins=3.64,19.82,27.92,40.06,56.25,72.44,88.63,104.81,201.94,404.28,808
.97
np.shape(bins)
```

Out[39]:
```
(11L,)
```

In [40]:
```
bins
```

Out[40]:
```
(3.64,
 19.82,
 27.92,
 40.06,
 56.25,
 72.44,
 88.63,
 104.81,
 201.94,
 404.28,
 808.97)
```

In [40]:
```

```

In [41]:
```
np.max(s_prob)
```

Out[41]:
```
0.00026189928029478283
```

In [42]:
```
checker=1,2,4,5,6,7,98,100,1257,2000
```

In [43]:
```
np.searchsorted(checker,7,'rightsided')
```

Out[43]:
```
6
```

In [44]:
```
def globalCalibrator(x):
    MU=x[0]
    SIGMA=x[1]
    RHO=x[2]
    VAR_s=SIGMA**2
    s_prob=np.zeros((s_n,1))
      # Redefine this every time
    for i in range(0,s_n):
        if i==0:
            d=(s_grid[1]-s_grid[0])/2
            s_prob[i]=lognorm_discrete(s_grid[i]-d,s_grid[i]+d,MU,VAR_
s)[0]
        else:
            d=(s_grid[i]-s_grid[i-1])/2
```

```
            s_prob[i]=lognorm_discrete(s_grid[i]-d,s_grid[i]+d,MU,VAR_
s)[0]
    XX=sum(s_prob)
    if XX<1:
        s_prob=s_prob+(1-XX)/s_n
    Xresult=so.minimize(calibration_eval,x0,args=a,method='Nelder-Mead
')   #Sloppy - relying on global definitions being just what want them
 to be
    KNn_US=Xresult.x[0]
    THETA=Xresult.x[1]
    pa_US=Xresult.x[2]
    w_US=(1-ALPHA)*A_US*KNn_US**ALPHA           # I'm not sure any of
this is necessary given how Python's local/global definitions are so f
luid
    r_US=ALPHA*A_US*KNn_US**(ALPHA-1)
    Ka_US=K_US-(KNn_US*(1-Na_US))
    KNa_US=Ka_US/Na_US
    YNn_US=A_US*(KNn_US**ALPHA)

    q_r_US=(rKa_qL_US**(-1))*Ka_US/LN_US
    q_US=q_r_US*r_US
    aBAR=(1/(pa_US*(1-PHI)))*((w_US*(1-XI)*Na_US/((1-GAMMA)))-PHI*(w_U
S*(1-XI)+q_US*LN_US+r_US*K_US))
    psi_s_vec1=THETA*((THETA*q_r_US/(1-THETA))**(RHO/(1-RHO)))*np.ones
((s_n,1))
    psi_s_vec2=(1-THETA)*np.power(s_grid,(RHO/(1-RHO)))
    psi_s_vec=psi_s_vec1+psi_s_vec2
    l_vec1=((pa_US*GAMMA*(1-THETA)*KAPPA_US*A_US/q_US)**(1/(1-GAMMA)))
    l_vec2=(np.power(psi_s_vec,((GAMMA-RHO)/(RHO*(1-GAMMA)))))
    l_vec3=(np.power(s_grid,(RHO/(1-RHO))))
    l_vec=l_vec1*np.multiply(l_vec2,l_vec3)
    k_l_vec=((THETA*q_r_US/(1-THETA))**(1/(1-RHO)))*((np.power(s_grid,
(-RHO/(1-RHO)))))
    k_vec = np.multiply(k_l_vec , l_vec)
    ya_vec=KAPPA_US*A_US*np.power((THETA*np.power(k_vec,RHO)+(1-THETA)
*np.power(np.multiply(s_grid,l_vec),RHO)),GAMMA/RHO)
    PI_vec=(1-GAMMA)*pa_US*ya_vec
    l_value = sum(np.multiply(s_prob,l_vec))           # Average farm
size
    YNa_value = sum(np.multiply(s_prob,ya_vec))        # Agricultural
Labor Productivity
    YN_value = Na_US*pa_US*YNa_value+(1-Na_US)*YNn_US  # Aggregate Lab
or Productivity
    binIndices=np.zeros((12,1))
    farms=np.zeros((12,1))
    lands=np.zeros((12,1))
    Capital=np.zeros((12,1))
    Output=np.zeros((12,1))
    ss=np.zeros((12,1))
    startIndex=0
    l_vec=np.array(l_vec)
    for i in range(0,11):
        binIndices[i]=(np.searchsorted(l_vec[:,0],bins[i],'rightsided'
```

```
))
        indexToUse=int(binIndices[i])
        farms[i]=np.sum(s_prob[startIndex:indexToUse])
        lands[i]=np.sum(np.multiply(l_vec[startIndex:indexToUse],s_pro
b[startIndex:indexToUse]))/l_value
        Capital[i]=np.sum(np.multiply(k_vec[startIndex:indexToUse],s_p
rob[startIndex:indexToUse]))
        Output[i]=np.sum(np.multiply(ya_vec[startIndex:indexToUse],s_p
rob[startIndex:indexToUse]))
        ss[i]=np.sum(np.multiply(s_grid[startIndex:indexToUse],s_prob[
startIndex:indexToUse]))
        startIndex=indexToUse+1
    farms[11]=np.sum(s_prob[startIndex:s_n])
    lands[11]=np.sum(np.multiply(l_vec[startIndex:s_n],s_prob[startInd
ex:s_n]))
    Capital[11]=np.sum(np.multiply(k_vec[startIndex:s_n],s_prob[startI
ndex:s_n]))
    Output[11]=np.sum(np.multiply(ya_vec[startIndex:s_n],s_prob[startI
ndex:s_n]))
    ss[11]=np.sum(np.multiply(s_grid[startIndex:s_n],s_prob[startIndex
:s_n]))
    kl_min_max_model=Capital[0]/(lands[0]*l_value)/(Capital[11]/(lands
[11]*l_value))
    farmdata=np.asmatrix(farm_pdf_data).T
    diff1=farms-farmdata
    diff2=np.asmatrix((kl_min_max_model-kl_min_max_data)/kl_min_max_da
ta)
    diff=np.vstack((diff1,diff2))
    print diff.T
    val=np.dot(diff1.T,diff1)
    print MU,SIGMA,RHO,val
    return val
```

In [45]:

```
xGuess=-5,1.9,.25
xGuess =-2.7889030987, 1.69147901341, 0.502734963506
optResult=so.minimize(globalCalibrator,xGuess,method='Nelder-Mead',tol
=1e-3)
```

```
[[ -2.67480427e-03   3.84390340e-03   6.76023481e-03  -5.19314851e-0
3
   -5.32757878e-03  -1.10751810e-02  -1.06718861e-03  -7.12350429e-0
4
    3.73819514e-03   3.66626832e-03  -1.80007770e-04   2.73850591e-0
3
    2.29730701e+03]]
-2.7889030987 1.69147901341 0.502734963506 [[ 0.00028224]]
[[  1.28039788e-02   2.02097517e-02   7.66724328e-03  -5.59007806e-0
3
   -6.88791732e-03  -1.28769938e-02  -2.79074878e-03  -2.30828490e-0
3
   -3.22831841e-03  -3.27133730e-03  -5.35342055e-03  -3.71165991e-0
3
    1.98646181e+03]]
-2.92834825364 1.69147901341 0.502734963506 [[ 0.0009523]]
```

```
[[  8.51199004e-03  -2.15877141e-03   3.24167164e-03  -9.07794597e-0
3
   -8.68851772e-03  -1.32134975e-02  -2.45211047e-03  -1.64030628e-0
3

    2.15940732e-03   4.99623475e-03   2.64350357e-03   1.03397908e-0
2
    3.09420168e+03]]
-2.7889030987 1.77605296408 0.502734963506 [[ 0.00057235]]
[[  1.74356401e-02  -1.04329818e-02   2.06763283e-04  -1.26062494e-0
2
   -1.02222042e-02  -1.41478541e-02  -2.58867361e-03  -2.04190435e-0
3
    3.35470371e-03   7.28845785e-03   5.40520020e-03   1.26685322e-0
2
    3.97606171e+03]]
-2.7889030987 1.69147901341 0.527871711681 [[ 0.00114142]]
[[ -5.56005968e-03   2.50463152e-02   1.13911518e-02  -9.15635009e-0
4
   -3.39307658e-03  -1.03578531e-02  -1.33576518e-03  -1.35476023e-0
3
   -2.01228587e-03  -4.55399384e-03  -6.60341493e-03  -5.44941013e-0
3
    1.46494821e+03]]
-2.88186653532 1.74786164719 0.477598215331 [[ 0.00100934]]
[[  2.63380878e-04   1.64893719e-02   8.67469864e-03  -4.04649180e-0
3
   -5.41948248e-03  -1.09941947e-02  -1.93496542e-03  -1.38962763e-0
3
   -1.61699495e-04  -1.47396844e-03  -3.79923260e-03  -1.45182539e-0
3
    1.87824878e+03]]
-2.85862567617 1.73376598875 0.490166589418 [[ 0.00053825]]
[[ -7.51518164e-03  -7.88603395e-03   4.95517361e-03  -7.17703643e-0
3
   -6.21327179e-03  -1.13029364e-02  -6.49194674e-04  -5.17485603e-0
4
    6.37094641e-03   8.02486210e-03   4.57841271e-03   1.19770977e-0
2
    2.84688198e+03]]
-2.69593966208 1.77605296408 0.494356047448 [[ 0.00063118]]
[[ -2.72616237e-03  -1.00699095e-03   5.71943040e-03  -6.82901696e-0
3
   -6.40617548e-03  -1.12855926e-02  -1.39313447e-03  -7.57279852e-0
4
    4.18855905e-03   5.25252012e-03   2.07944360e-03   7.80616770e-0
3
    2.58878122e+03]]
-2.75404180997 1.75490947641 0.496450776462 [[ 0.0003691]]
[[ -1.18923479e-02   1.48399066e-02   1.08871457e-02  -1.15554058e-0
3
   -2.57337177e-03  -8.93610528e-03  -4.76391386e-04  -2.88927979e-0
4
    2.84343911e-03  -4.98623825e-04  -4.20202154e-03  -3.92751060e-0
```

```
3
    1.63830416e+03]]
-2.81214395786 1.67738335496 0.490166589418 [[ 0.00060972]]
[[  3.43408790e-03   2.10188373e-03   4.90945747e-03  -7.33814706e-0

3
   -6.56579660e-03  -1.27630450e-02  -1.64976646e-03  -1.01694025e-0
3
    2.11524244e-03   3.81497152e-03   1.03894624e-03   6.56809631e-0
3
    2.62345209e+03]]
-2.79471331349 1.7513855618 0.499592869984 [[ 0.00036717]]
[[ -1.88280579e-03  -1.21192319e-02   2.38576263e-03  -9.09197609e-0
3
   -7.48109293e-03  -1.22657070e-02  -1.17677132e-03  -5.92923309e-0
4
    7.03108548e-03   9.58721863e-03   6.23724438e-03   1.38275685e-0
2
    3.35097656e+03]]
-2.6998131386 1.73141671234 0.50901915055 [[ 0.00081838]]
[[ -4.94388741e-05   8.57703606e-03   7.78625182e-03  -5.88535705e-0
3
   -4.87562428e-03  -1.21322326e-02  -1.23868536e-03  -1.26629079e-0
3
    1.63317573e-03   1.58190024e-03  -1.50623699e-03   2.05307004e-0
3
    2.16002052e+03]]
-2.81892254178 1.73317866964 0.494879729701 [[ 0.00035459]]
[[  3.29037340e-03   1.14627467e-02   7.01405970e-03  -5.49952120e-0
3
   -5.46575346e-03  -1.19435931e-02  -1.31045361e-03  -1.47943765e-0
3
    4.77182599e-04   5.18419907e-04  -2.33334917e-03  -1.38645892e-0
4
    2.14495681e+03]]
-2.84765082601 1.69578602016 0.501687598999 [[ 0.00040405]]
[[ -1.35806931e-03   2.42236155e-03   5.24099317e-03  -6.15611009e-0
3
   -5.98891777e-03  -1.17278910e-02  -7.74760171e-04  -1.33810923e-0
3
    3.65064148e-03   3.91354186e-03   1.03072828e-03   5.71413082e-0
3
    2.47090239e+03]]
-2.77744406398 1.74012861235 0.497759982096 [[ 0.00031124]]
[[ -6.14728396e-03   7.52671472e-03   8.10406028e-03  -3.86978446e-0
3
   -4.17247692e-03  -1.00856142e-02  -7.82256439e-04  -4.73854230e-0
4
    3.44930501e-03   1.99394571e-03  -1.46185935e-03   4.86989418e-0
4
    2.03312092e+03]]
-2.79546648948 1.6918053018 0.497323580218 [[ 0.0003133]]
[[ -6.74763290e-03   4.46394235e-04   6.20922984e-03  -5.41967736e-0
3
```

```
   -4.61650491e-03  -1.04109697e-02  -4.91849419e-04  -2.12056711e-0
4
    5.28277416e-03   5.23073020e-03   1.28248400e-03   3.90811627e-0
3

    2.35940244e+03]]
-2.755619893 1.68242994873 0.503665954179 [[ 0.00031583]]
[[ -4.91706325e-03   2.39011966e-03   6.45373900e-03  -5.35750632e-0
3
   -4.67089363e-03  -1.09999989e-02  -3.80943932e-04  -6.24949058e-0
4
    4.65351625e-03   3.92118998e-03   5.48907885e-04   3.49996742e-0
3
    2.30340136e+03]]
-2.77144555519 1.69511712896 0.50146939806 [[ 0.00029318]]
[[  1.00345312e-04  -1.84054616e-03   4.43631050e-03  -7.47416832e-0
3
   -6.48897309e-03  -1.16515978e-02  -1.32740212e-03  -1.34110010e-0
3
    4.60322089e-03   5.55722893e-03   2.35125278e-03   7.61892765e-0
3
    2.74101948e+03]]
-2.7630619891 1.72601120134 0.503985982223 [[ 0.00037602]]
[[ -4.66269114e-03   5.41173990e-03   7.01097256e-03  -4.94789359e-0
3
   -4.42302486e-03  -1.07833673e-02  -7.58767956e-04  -4.10911723e-0
4
    3.21195385e-03   3.15021724e-03  -4.02239698e-04   2.16496242e-0
3
    2.19123729e+03]]
-2.78736536438 1.70035677669 0.498989180719 [[ 0.00028634]]
[[ -7.01181674e-03   5.91810926e-03   7.87954154e-03  -3.89101078e-0
3
   -3.57272960e-03  -1.05705378e-02  -2.94822997e-04  -1.81487436e-0
4
    4.36643817e-03   3.10996411e-03  -1.17826725e-03  -1.41962904e-0
4
    2.08088107e+03]]
-2.78769861487 1.65117333369 0.504369046094 [[ 0.00031618]]
[[ -2.81275722e-03   3.03735474e-03   6.19594937e-03  -5.77746320e-0
3
   -5.08147656e-03  -1.17571801e-02  -3.63574458e-04  -1.06281769e-0
3
    3.80657014e-03   3.58076190e-03   6.10905360e-04   4.20485264e-0
3
    2.36827863e+03]]
-2.7800077017 1.71788979268 0.499412248096 [[ 0.00029959]]
[[ -5.48931145e-03   4.62691033e-03   7.52281906e-03  -4.30948575e-0
3
   -4.53215372e-03  -1.05512685e-02  -7.16556762e-04  -4.68740258e-0
4
    4.23628583e-03   3.50807391e-03  -6.92277003e-04   1.34726141e-0
3
    2.16782683e+03]]
```

-2.78513497715 1.67341215335 0.502716780094 [[0.00029186]]
[[-3.42631818e-03   7.13294401e-03   7.30332247e-03 -4.73468171e-0
3
   -4.40927450e-03 -1.10229570e-02 -5.28069967e-04 -8.25279798e-0
4
    3.31640270e-03   2.08067353e-03 -1.15206869e-03   7.88784101e-0
4
    2.12999296e+03]]
-2.80282340497 1.68171483334 0.501491218154 [[0.00029756]]
[[-4.54810775e-03   3.56755772e-03   6.66653020e-03 -5.20001435e-0
3
   -4.60265201e-03 -1.10031140e-02 -4.15445010e-04 -6.73118134e-0
4
    4.32604356e-03   3.46490528e-03   1.23633440e-04   2.81154024e-0
3
    2.25845894e+03]]
-2.77929001763 1.69176655505 0.501474853083 [[0.00028641]]
[[-2.56388082e-03   3.81051576e-03   6.33418176e-03 -5.67569005e-0
3
   -5.03788845e-03 -1.17597047e-02 -3.86537838e-04 -1.09417916e-0
3
    3.59351975e-03   3.28370868e-03   3.32629546e-04   3.74557284e-0
3
    2.33712104e+03]]
-2.78523734333 1.71565607675 0.499415884778 [[0.00029628]]
[[-4.67494421e-03   4.51254339e-03   7.05687612e-03 -4.82654478e-0
3
   -4.32970605e-03 -1.08671913e-02 -9.30527287e-04 -6.19335092e-0
4
    4.31229706e-03   3.16599021e-03 -2.75318850e-04   1.98217308e-0
3
    2.20602005e+03]]
-2.78516056869 1.6839731342 0.501891556265 [[0.00028603]]
[[-3.46484420e-03   6.05444467e-03   6.78311609e-03 -5.22916690e-0
3
   -4.76733012e-03 -1.05893930e-02 -6.42650222e-04 -8.82176648e-0
4
    3.39779193e-03   2.78915543e-03 -6.92518942e-04   1.78291201e-0
3
    2.20427142e+03]]
-2.79499600355 1.69210606114 0.500935613911 [[0.00028105]]
[[-2.61678768e-03   6.99380174e-03   6.83897053e-03 -4.57416262e-0
3
   -4.88066260e-03 -1.13328241e-02 -7.40055639e-04 -9.71400770e-0
4
    3.00437079e-03   2.39723651e-03 -9.86975505e-04   1.41635496e-0
3
    2.16731136e+03]]
-2.80284899651 1.69227581419 0.500665994324 [[0.00029495]]
[[-2.62849992e-03   4.28682501e-03   6.72058283e-03 -5.22018824e-0
3
   -5.41645859e-03 -1.06876107e-02 -8.08300013e-04 -1.08094721e-0
3

```
    3.84285525e-03   3.58176651e-03  -2.30070173e-04   2.11993136e-0
3
    2.28304151e+03]]
-2.79200774958 1.67801536248 0.504718908402 [[ 0.00027523]]

[[ -1.73470314e-03   3.85741078e-03   6.57226729e-03  -6.04228591e-0
3
   -4.91262767e-03  -1.09661948e-02  -1.12613540e-03  -8.55141991e-0
4
    4.09276506e-03   3.47017088e-03   5.85285232e-05   2.02506076e-0
3
    2.33443666e+03]]
-2.79432894218 1.66684465538 0.507583772243 [[ 0.00027688]]
[[ -8.50405970e-04   4.83724517e-03   6.23388563e-03  -5.82563168e-0
3
   -5.35107591e-03  -1.11311994e-02  -1.13176614e-03  -7.78052942e-0
4
    3.00964015e-03   3.18306711e-03  -2.65607681e-04   2.58564620e-0
3
    2.30684453e+03]]
-2.79877733253 1.69042715716 0.50370143428 [[ 0.00027729]]
[[ -1.95911057e-03   6.07612596e-03   6.61464338e-03  -5.43414494e-0
3
   -5.03694664e-03  -1.09506794e-02  -1.02849250e-03  -7.19748350e-0
4
    3.40514183e-03   2.73505884e-03  -7.83862086e-04   1.58870962e-0
3
    2.23217421e+03]]
-2.80161762507 1.68222004045 0.503502340889 [[ 0.00028312]]
[[ -2.34460455e-03   4.08357286e-03   6.88765423e-03  -5.08428956e-0
3
   -5.26169601e-03  -1.10523691e-02  -1.06565322e-03  -7.21838853e-0
4
    3.62078980e-03   3.20646884e-03  -2.74877030e-04   2.52107365e-0
3
    2.27527230e+03]]
-2.79208173029 1.68916427017 0.502926807852 [[ 0.00027678]]
[[ -7.00699298e-04   3.47477687e-03   6.00578237e-03  -5.97808593e-0
3
   -5.47153754e-03  -1.13303833e-02  -1.36747528e-03  -4.71938812e-0
4
    3.61806052e-03   3.71542876e-03   7.23776326e-05   2.90316547e-0
3
    2.38622820e+03]]
-2.79358187138 1.67963179873 0.506629153112 [[ 0.00028011]]
[[ -1.47764254e-03   3.87779143e-03   6.52907186e-03  -5.45143200e-0
3
   -5.63249473e-03  -1.14506186e-02  -8.77393075e-04  -5.75153662e-0
4
    3.66504494e-03   3.31750312e-03  -2.04304568e-05   2.58311744e-0
3
    2.34260840e+03]]
-2.79393540442 1.68275036433 0.505205768312 [[ 0.00028462]]
[[ -2.48615761e-03   4.51232139e-03   6.47684474e-03  -5.49017940e-0
```

```
[[
3
   -5.00069804e-03  -1.08706182e-02  -9.37403903e-04  -6.27635641e-0
4
    3.45859930e-03   3.39364533e-03  -2.52270660e-04   2.32074763e-0

3
    2.27918119e+03]]
-2.79204473994 1.68358981633 0.503822858127 [[ 0.00027201]]
[[ -1.73896131e-03   4.56532689e-03   6.47673940e-03  -5.52388021e-0
3
   -5.05957149e-03  -1.15406016e-02  -9.53619438e-04  -6.39883849e-0
4
    3.42454357e-03   3.38189856e-03  -2.47548569e-04   2.35307956e-0
3
    2.29488660e+03]]
-2.79539254105 1.68422125982 0.504210171341 [[ 0.00028519]]
[[ -3.04904409e-03   4.84774104e-03   7.07931751e-03  -4.88682603e-0
3
   -5.10428521e-03  -1.09616829e-02  -1.01360644e-03  -6.92311845e-0
4
    3.61969893e-03   3.05605965e-03  -3.35919851e-04   1.95043695e-0
3
    2.24356398e+03]]
-2.79350187657 1.68506071181 0.502827261156 [[ 0.00028087]]
[[ -3.49729373e-03   4.76177767e-03   6.60407297e-03  -5.32519321e-0
3
   -4.83601645e-03  -1.07524979e-02  -8.51908185e-04  -5.63283009e-0
4
    4.03835723e-03   3.30445328e-03  -3.93474438e-04   2.00234861e-0
3
    2.23531214e+03]]
-2.789643703 1.68022266726 0.503369180449 [[ 0.00027832]]
[[ -2.89638172e-03   3.73974630e-03   6.77666283e-03  -5.12713897e-0
3
   -5.30938295e-03  -1.12180537e-02  -7.05969832e-04  -4.45440625e-0
4
    4.02734891e-03   3.44499108e-03  -6.65816553e-05   2.25102424e-0
3
    2.29540731e+03]]
-2.78896225178 1.67615785223 0.505113370162 [[ 0.00028248]]
[[ -2.85867556e-03   4.91037777e-03   6.51246139e-03  -5.45402751e-0
3
   -4.97214870e-03  -1.08545572e-02  -9.28484899e-04  -6.22844760e-0
4
    3.85759687e-03   3.21655463e-03  -4.03853669e-04   2.09620366e-0
3
    2.25100203e+03]]
-2.79236697037 1.68283499692 0.503398788408 [[ 0.00027802]]
[[ -1.71223417e-03   4.27149050e-03   6.53538346e-03  -5.45149790e-0
3
   -5.63868079e-03  -1.08446259e-02  -9.19598333e-04  -6.14948776e-0
4
    3.48741995e-03   3.13600745e-03  -1.60231266e-04   2.40363311e-0
3
```

In [45]:

In [46]:
```
optResult
```

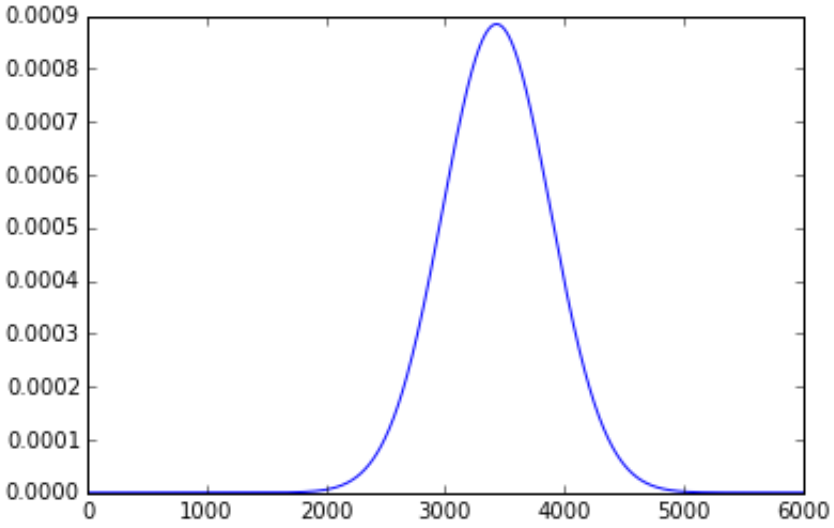
Out[46]:
```
 status: 0
   nfev: 63
success: True
    fun: 0.00027174323130965163
      x: array([-2.79013625,  1.68176697,  0.50379805])
message: 'Optimization terminated successfully.'
    nit: 31
```

In [47]:
```
MUHat=optResult.x[0]
SIGMAHat=optResult.x[1]
RHOHat=optResult.x[2]
RHOHat,MUHat,SIGMAHat
```

Out[47]:
```
(0.50379805395728861, -2.7901362478963154, 1.6817669666255064)
```

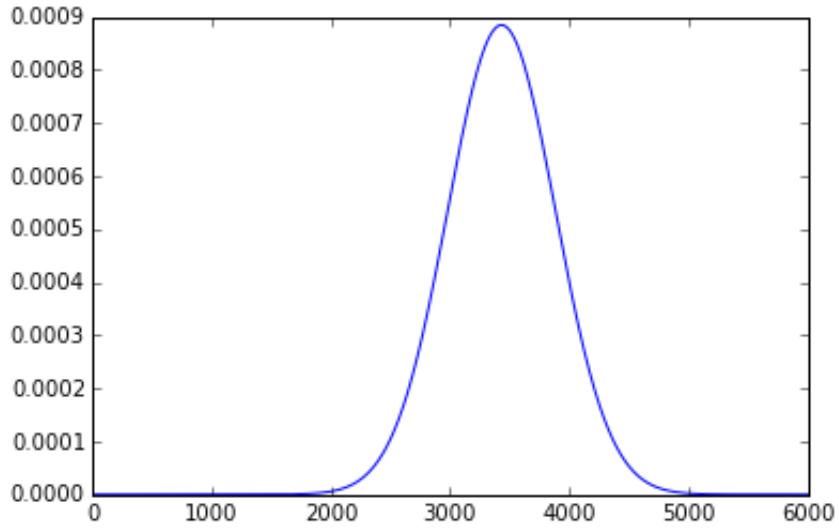
In [48]:
```
s_probHat=np.zeros((s_n,1))
for i in range(0,s_n-1):
    if i==0:
        d=(s_grid[1]-s_grid[0])/2
        s_probHat[i]=lognorm_discrete(s_grid[i]-d,s_grid[i]+d,MUHat,SI
GMAHat)[0]
    else:
        d=(s_grid[i]-s_grid[i-1])/2
        s_probHat[i]=lognorm_discrete(s_grid[i]-d,s_grid[i]+d,MUHat,SI
GMAHat)[0]
```

In [49]:
```
plt.plot(s_probHat)
plt.show()
```



In [50]:
```
XX=sum(s_probHat)
if XX<1:
    s_probHat=s_probHat+(1-XX)/s_n
```

In [51]:
```
plt.plot(s_probHat)
plt.show()
print XX
```




```
0.998562706272
```

It seems like a good idea to make a little code that can generate quantities of interest based on this. So, here goes:

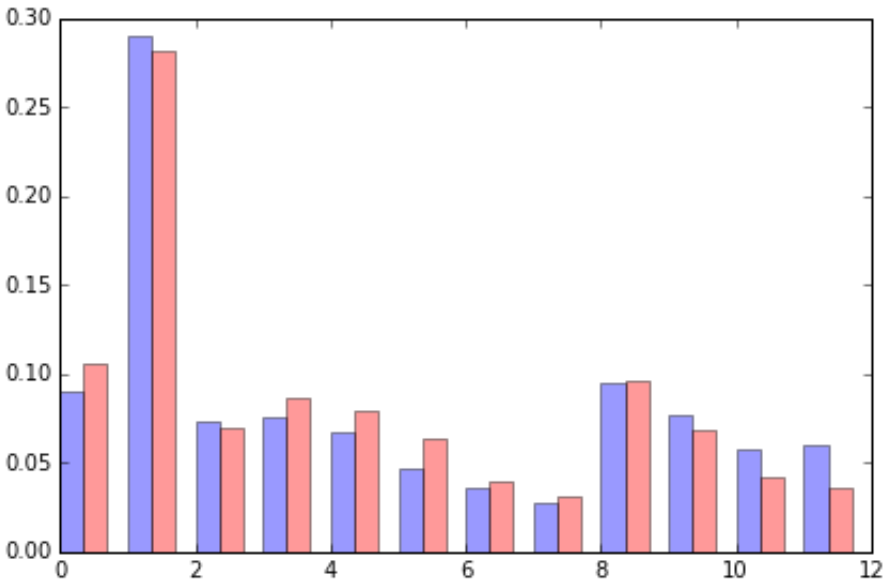
In [52]:
```
    VAR_sHAT=SIGMA**2
    XresultHat=so.minimize(calibration_eval,x0,args=a,method='Nelder-M
ead')   #Sloppy - relying on global definitions being just what want t
hem to be
    KNn_US=XresultHat.x[0]
    THETA=XresultHat.x[1]
    pa_US=XresultHat.x[2]
    w_US=(1-ALPHA)*A_US*KNn_US**ALPHA           # I'm not sure any of
this is necessary given how Python's local/global definitions are so f
luid
    r_US=ALPHA*A_US*KNn_US**(ALPHA-1)
    Ka_US=K_US-(KNn_US*(1-Na_US))
    KNa_US=Ka_US/Na_US
    YNn_US=A_US*(KNn_US**ALPHA)

    q_r_US=(rKa_qL_US**(-1))*Ka_US/LN_US
    q_US=q_r_US*r_US
    aBAR=(1/(pa_US*(1-PHI)))*((w_US*(1-XI)*Na_US/((1-GAMMA)))-PHI*(w_U
S*(1-XI)+q_US*LN_US+r_US*K_US))
    psi_s_vec1=THETA*((THETA*q_r_US/(1-THETA))**(RHO/(1-RHO)))*np.ones
((s_n,1))
    psi_s_vec2=(1-THETA)*np.power(s_grid,(RHO/(1-RHO)))
    psi_s_vec=psi_s_vec1+psi_s_vec2
    l_vec1=((pa_US*GAMMA*(1-THETA)*KAPPA_US*A_US/q_US)**(1/(1-GAMMA)))
    l_vec2=(np.power(psi_s_vec,((GAMMA-RHO)/(RHO*(1-GAMMA)))))
    l_vec3=(np.power(s_grid,(RHO/(1-RHO))))
```

```
    l_vec=l_vec1*np.multiply(l_vec2,l_vec3)
    k_l_vec=((THETA*q_r_US/(1-THETA))**(1/(1-RHO)))*((np.power(s_grid,
(-RHO/(1-RHO)))))
    k_vec = np.multiply(k_l_vec , l_vec)
    ya_vec=KAPPA_US*A_US*np.power((THETA*np.power(k_vec,RHO)+(1-THETA)
*np.power(np.multiply(s_grid,l_vec),RHO)),GAMMA/RHO)
    PI_vec=(1-GAMMA)*pa_US*ya_vec
    l_value = sum(np.multiply(s_probHat,l_vec))           # Average fa
rm size
    YNa_value = sum(np.multiply(s_probHat,ya_vec))        # Agricultur
al Labor Productivity
    YN_value = Na_US*pa_US*YNa_value+(1-Na_US)*YNn_US  # Aggregate Lab
or Productivity
    binIndices=np.zeros((12,1))
    farms=np.zeros((12,1))
    lands=np.zeros((12,1))
    Capital=np.zeros((12,1))
    Output=np.zeros((12,1))
    ss=np.zeros((12,1))
    startIndex=0
    l_vec=np.array(l_vec)
    for i in range(0,11):
        binIndices[i]=(np.searchsorted(l_vec[:,0],bins[i],'rightsided'
))
        indexToUse=int(binIndices[i])
        farms[i]=np.sum(s_prob[startIndex:indexToUse])
        lands[i]=np.sum(np.multiply(l_vec[startIndex:indexToUse],s_pro
b[startIndex:indexToUse]))/l_value
        Capital[i]=np.sum(np.multiply(k_vec[startIndex:indexToUse],s_p
rob[startIndex:indexToUse]))
        Output[i]=np.sum(np.multiply(ya_vec[startIndex:indexToUse],s_p
rob[startIndex:indexToUse]))
        ss[i]=np.sum(np.multiply(s_grid[startIndex:indexToUse],s_prob[
startIndex:indexToUse]))
        startIndex=indexToUse+1
    farms[11]=np.sum(s_prob[startIndex:s_n])
    lands[11]=np.sum(np.multiply(l_vec[startIndex:s_n],s_prob[startInd
ex:s_n]))
    Capital[11]=np.sum(np.multiply(k_vec[startIndex:s_n],s_prob[startI
ndex:s_n]))
    Output[11]=np.sum(np.multiply(ya_vec[startIndex:s_n],s_prob[startI
ndex:s_n]))
    ss[11]=np.sum(np.multiply(s_grid[startIndex:s_n],s_prob[startIndex
:s_n]))
    kl_min_max_model=Capital[0]/(lands[0]*l_value)/(Capital[11]/(lands
[11]*l_value))
```

In [53]:
```
print kl_min_max_model
print farms.T
print farm_pdf_data
```

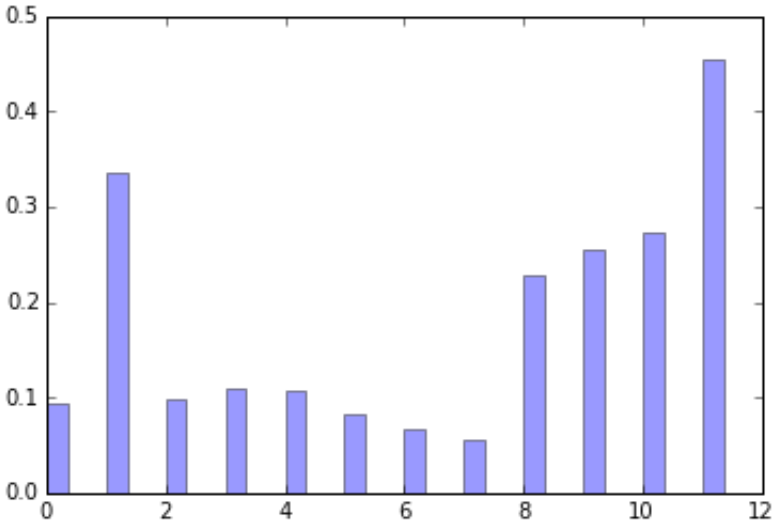
```
[ 2855.81567799]
[[ 0.09009124  0.29068376  0.07367146  0.07601274  0.06771613  0.04721
292
   0.03536576  0.02764707  0.09460149  0.07721176  0.05791016  0.05945
275]]
[0.1056, 0.2813, 0.0698, 0.0871, 0.0794, 0.0633, 0.0397, 0.031, 0.0964
, 0.0679, 0.042, 0.0365]
```

In [54]:
```
n_groups = 12

fig, ax = plt.subplots()

index = np.arange(n_groups)
bar_width = 0.35

opacity = 0.4
error_config = {'ecolor': '0.3'}

rects1 = plt.bar(index, np.array(farms), bar_width,
                 alpha=opacity,
                 color='b',
                 label='Model')

rects2 = plt.bar(index + bar_width, np.array(farm_pdf_data), bar_width
,
                 alpha=opacity,
                 color='r',
                 label='Data')

plt.tight_layout()
plt.show()
```



That is a pretty close fit. What about the other stuff?

In [55]:
```
fig, ax = plt.subplots()
rects1 = plt.bar(index, np.array(Capital), bar_width,
                 alpha=opacity,
                 color='b',
                 label='Model')
plt.show()
```



In [55]: