Modern Value Chains and the Organization of Agrarian Production

June 30, 2015

Abstract

Empirical studies of agrarian production in developing countries have historically found that smallholders possess a productivity advantage over larger farms. Eswaran and Kotwal (1986) famously derive this inverse farm-size/productivity relationship from the structure of agrarian production. The focal prediction of their model is that, in otherwise equivalent economies, a more egalitarian land distribution raises output and producer welfare. The traditional (spot) procurement system implicit in the Eswaran and Kotwal model, however, diverges fundamentally from modern (contractual) procurement practices. We therefore develop a new model of agrarian production and ask how the welfare effects of land redistribution change with the introduction of such modern value chains. In our model, the inverse farm-size/productivity relationship persists, but the effects of land redistribution change.

Keywords: smallholder agriculture, agricultural value chains, contract farming, land reform

JEL Class: Q15, C63, 013, Q13, D23

1 Introduction

Population growth, rising incomes, and continued urbanization are increasing food demand in developing economies. Accordingly, FAO (2011) predicts that food prices will become substantially higher and more volatile unless global agricultural production nearly doubles by mid-century. Due to continuing land scarcity, lackluster productivity growth, and little remaining farm price repression, the policy options for increasing agricultural production in developing countries are limited. However, due to the often-observed productivity advantage of small-scale agricultural producers in developing countries, redistributive land reform has been proposed not only as a means to increase equity and reduce poverty, but also as a means to improve agricultural productivity (Eswaran and Kotwal, 1986; Lipton, 2009; Keswell and Carter, 2014).[1]

Eswaran and Kotwal (1986, hereafter EK) famously predict that, in otherwise equivalent economies, more egalitarian land distributions raise output and producer welfare. Adopting their terminology, EK's key result is that greater "equity" also improves "efficiency." As a model of modern agrarian economies, however, the EK model has a substantial shortcoming: each producer faces a single output market, which is implicitly an informal spot market. Barrett et al. (2012) show that this institutional arrangement is no longer descriptive of many developing countries. The growth of export horticulture, the diffusion of supermarkets, and the proliferation of grades and standards have led to the modernization of agricultural value chains. Contract farming has become central to modern procurement systems, but scale-biased participation raises questions as to the contemporary policy relevance of redistributive measures. (See section 2.2 for details.)

We therefore re-examine the equity-efficiency relationship in a model better suited to the new agricultural economy. Specifically, we extend the EK model by adding a modern agricultural value chain, as described by Barrett et al. (2012). While the inverse farm-size/productivity relationship persists in our model, we find that a more egalitarian land distribution leads to non-monotonic changes in key model outcomes (e.g., producer welfare). In this sense, we find an eventual "equity-efficiency" tradeoff in the distribution of land. Our results thus suggest that the case for redistributive measures has been diminished by the emergence of modern agricultural value chains. In what follows, section 2 provides relevant background information, section 3 presents our model of the new agricultural economy, section 4 discusses results and sensitivity analysis, and section 5 concludes.

[1] Lipton (2009, p.328) defines redistributive land reform as "legislation intended and likely to directly redistribute ownership of, claims on, or rights to current farmland, and thus to benefit the poor by raising their absolute and relative status, power, and/or income, compared with likely situations without the legislation."

2 Background

This section comprises two short and fairly independent subsections. The first subsection discusses EK in the context of the empirical and theoretical work on the relationship between farm size and productivity. The second subsection motivates our theoretical model by considering the recent restructuring of global agri-food systems and the rise of modern value chains.

2.1 Farm Size and Productivity

Empirical studies of agrarian production in developing countries have historically found that smallholders possess a productivity advantage over larger farms. The Indian Ministry of Food and Agriculture's *Studies in the Economics of Farm Management* (SEFM) was among the first to document an inverse relationship between farm size and land productivity (Sen, 1962). In the 1970s and 1980s, a number of influential cross-country studies observed this inverse relationship across Africa, Asia, and Latin America (Barraclough, 1973; Berry and Cline, 1979; Cornia, 1985). More recent empirical studies have investigated the relationship between farm size and profitability, and have also found a robust negative relationship (van Zyl, Binswanger, and Thirtle, 1995; Heltberg, 1998; Deininger, Zegarra, and Lavadenz, 2003).[2]

Labor market imperfections have occupied a central role in the leading explanations of the inverse relationship.[3] In his "dual labor cost theory," Sen (1966) contended that there exists a gap between the real cost of labor in peasant farming and the market wage rate. Since capitalist farms face higher equilibrium labor costs, peasant farms use labor more intensively and witness greater land productivity. Bhagwati and Chakravarty (1969), however, found that the inverse relationship persists when examining only capitalist farms. Further, Feder (1985) argued that multiple factor market imperfections must be present to generate a systematic relationship between farm size and productivity. Building on the "dual labor cost theory," EK influentially incorporated such considerations into a revised theoretical model of the inverse relationship.

Drawing on the seminal work of Roemer (1982), EK developed a single-period model of class formation in an agrarian economy characterized by labor-market and credit-market imperfections. In the model, hired labor requires supervision due to moral hazard, and access to credit largely depends on the amount of land an agent owns. Given such imperfections, EK found land-to-labor ratios to be increasing in land

[2]Other widely-cited empirical studies include Yotopoulos and Lau (1973), Carter (1984), Bhalla and Roy (1988), Assunção and Braido (2007), and Barrett, Bellemare, and Hou (2010). While a number of the empirical studies have been challenged as reflecting statistical artifacts, these challenges have in turn been challenged. See, for example, Carter (1984), Barrett, Bellemare, and Hou (2010), and Carletto, Savastano, and Zezza (2013). For detailed reviews of the controversy, see Binswanger, Deininger, and Feder (1995) and Eastwood, Lipton, and Newell (2010).

[3]A number of other explanations have been put forth (e.g., decreasing returns to scale, land quality heterogeneity, and differential responses to uncertainty), but much of the empirical work supports the labor market imperfections hypothesis (Barraclough, 1973; Berry and Cline, 1979; Carter, 1984; Cornia, 1985; Binswanger, Deininger, and Feder, 1995; van Zyl, Binswanger, and Thirtle, 1995; Heltberg, 1998; Deininger, Zegarra, and Lavadenz, 2003). See Henderson (2015) for a detailed review of the literature.

endowments, which implies an inverse relationship between land productivity and ownership landholdings. Correspondingly, EK demonstrated that a more egalitarian distribution of land ownership increases aggregate agricultural production, enhances overall welfare, and reduces poverty. Even the landless were found to benefit from redistributive measures, due to increased labor demand and a corresponding rise in wages.

EK assume that each producer faces a single output market, which is implicitly an informal spot market. The recent radical restructuring of global agri-food systems raises important questions about this assumption, since such traditional (spot) procurement systems diverge fundamentally from modern (contractual) procurement practices. Specifically, economies of scale in finance and access to capital imply scale-variant grower capacity to meet the demands of modern value chains. Further, scale-invariant contract-related transaction costs suggest that larger-scale private trading and marketing could reduce costs (Collier and Dercon, 2009). As discussed in the next subsection, scale-biased participation in modern contractual farming arrangements may alter predictions about the welfare benefits from redistributive land reform.

2.2 Modern Value Chains

Since the early 1980s, global agri-food systems have undergone a continual and fundamental transformation, especially in developing countries. Developments on the demand side (e.g., rising incomes and increasing urbanization) and supply side (e.g., foreign direct investment and changing technology) stimulated growth in export horticulture, the dissemination of supermarkets, and the proliferation of grades and standards (McCullough, Pingali, and Stamoulis, 2008; Reardon, Timmer, and Berdegué, 2008; Eastwood, Lipton, and Newell, 2010). The diffusion of supermarkets has been particularly dramatic. For example, Reardon and Berdegué (2002) found that supermarket shares in Latin America increased from 10–20 percent of national retail sales in 1990 to 50–60 percent by 2000. Reardon et al. (2003) observed comparable patterns in eastern and southeastern Asia (e.g., Taiwan, the Philippines, and the Republic of Korea) as well as southern and eastern Africa (e.g., South Africa and Kenya). Dries, Reardon, and Swinnen (2004) documented similar changes in central and eastern Europe (e.g., Czech Republic, Hungary, and Poland).

Traditional wholesalers and brokers in developing nations typically rely on informal, spot transactions. However, the restructuring of agricultural output markets has commonly supported the augmented quality standards of downstream entities (e.g., supermarkets and export firms) through parallel modern procurement systems (McCullough, Pingali, and Stamoulis, 2008; Reardon et al., 2009). These parallel systems often include specialized wholesalers, centralized rather than per-store procurement, the standardization and harmonization of product and delivery attributes, and a reliance on contract farming (Reardon et al., 2003; Reardon, Timmer, and Berdegué, 2008; Barrett et al., 2012). The modern-sector prevalence of con-

tract farming diverges fundamentally from traditional-sector practices.[4] Empirical studies of the effects of such arrangements suggest that they can raise grower welfare and enhance rural development by increasing productivity, profitability, and employment.[5] However, in many scenarios these gains appear quite limited (Singh, 2002; Sivramkrishna and Jyotishi, 2008; Escobal and Cavero, 2011; Barrett et al., 2012).

Two common features of contractual farming arrangements can limit producer gains: the monopsony power of procuring firms, and the exclusion of smaller-scale producers. Contract farming in developing-country agriculture is frequently characterized by monopsonistic or oligopsonistic competition: a single large buyer (or possibly a few buyers) chooses the terms (e.g., prices, quantities, or quality) of contracts available to numerous sellers. The rapid consolidation of supermarkets in developing countries is illustrative. In Latin America, on average approximately two-thirds of the supermarket sector is controlled by the top five chains, with some particularly high concentrations in Central America (Reardon and Berdegué, 2002).[6] Similar patterns have been documented in Africa, Asia, as well as central and eastern Europe (Neven et al., 2009; Hu et al., 2004; Dries, Reardon, and Swinnen, 2004). This consolidation of downstream segments raises concerns that the potential benefits of contract farming are restricted by asymmetric power in the negotiation of contractual terms (Sivramkrishna and Jyotishi, 2008).

Additional concerns arise as agro-industrial firms often eschew contracting with smaller-scale, less capital-intensive producers (Barrett et al., 2012). In Africa, such exclusion has been observed in Ghana (Trienekens and Willems, 2007), Kenya (Rao and Qaim, 2011), Senegal (Maertens and Swinnen, 2009), South Africa (Trienekens and Willems, 2007), and Uganda (Bolwig, Gibbon, and Jones, 2009). Asian examples include China (Stringer, Sang, and Croppenstedt, 2009), India (Singh, 2002), and Indonesia (Simmons, Winters, and Patrick, 2005). In Latin America, exclusion has been documented in Brazil (Farina, 2002), Costa Rica (Alvarado and Charmel, 2002), Guatemala (Hernández, Reardon, and Berdegué, 2007), Mexico (Key and Runsten, 1999), Nicaragua (Michelson, Reardon, and Perez, 2011), and Peru (Escobal and Cavero, 2011). A few studies document cases of smallholder *inclusion*, but this is rare and usually reflects special circumstances. For example, outside assistance is occasionally provided to a procuring firm or contracted producers as a result of partnerships between public-sector and private-sector stakeholders, as has been observed in Kenya, South Africa, Thailand, and Zimbabwe (Boselie, Henson, and Weatherspoon, 2003).

Scale-biased participation can arise from scale-variant grower capacity to meet requisite standards or scale-invariant contract-related transaction costs. Scale-variant capacity to meet modern-sector standards

[4]Singh (2002, p.1621) defines contract farming as "a system for the production and supply of agricultural produce under forward contracts, the essence of such contracts being a commitment to provide an agricultural commodity of a type, at a time and a price, and in the quantity required by a known buyer."

[5]See, for example, Warning and Key (2002), Simmons, Winters, and Patrick (2005), Bolwig, Gibbon, and Jones (2009), Minten, Randrianarison, and Swinnen (2009), and Miyata, Minot, and Hu (2009).

[6]In Guatemala, Costa Rica, and El Salvador these figures reach as high as 99, 96, and 85 percent, respectively.

can be understood in terms of grower-side incentives and constraints. On the one hand, most costs associated with modern-sector production are fixed and "up front." Examples include the cost of information search (e.g., learning to grow crops of the desired shape, flavor, or variety), physical capital investment (e.g., irrigation technology), certification (e.g., EurepGAP/GLOBALGAP), and collective action (e.g., forming producer cooperatives).[7] On the other hand, benefits generally accrue post-harvest. The principal motivation for selling in high-value markets is the ultimate receipt of output price premiums, which may arise from product quality differences, compliance incentives, or the ability to capture a greater part of the marketing margin (Minten, Randrianarison, and Swinnen, 2009; Neven et al., 2009).[8] Producers often require credit to overcome this temporal mismatch, but cash-strapped smallholders typically have limited access to formal and informal lending institutions (Stiglitz and Weiss, 1981; Carter, 1988; Santos and Barrett, 2011).

Scale-biased participation also arises from contract-related transaction costs, which can be understood in terms of procurer-side incentives. An agro-industrial firm pursues contractual farming arrangements in order to minimize transaction, production, and management costs across available alternatives (Herath and Weersink, 2009). Predominant costs include the search for prospective growers, the screening of those growers, the negotiation of contracts, the transfer of goods, services, or property rights, the monitoring of grower behavior, and the enforcement of the terms of the contract (Key and Runsten, 1999). These costs are largely independent of the scale of the grower, thereby creating an incentive for the procuring firm to increase the average scale of contracted producers. Procurer-side incentives can therefore be a critical determinant of grower contracting opportunities (Key and Runsten, 1999; Simmons, Winters, and Patrick, 2005).

In summary, this subsection has reviewed a number of "stylized" facts about agricultural value chains in developing economies. First, modern value chains commonly exist in parallel to traditional channels. Second, modern channels are frequently characterized by monopsonistic procurers. Third, in order to reap the benefits of modern-sector participation (e.g., output price premiums), prospective growers typically must bear the burden of substantial fixed costs—an obstacle that credit-constrained smallholders find difficult to overcome. Finally, downstream entities in the modern sector typically incur fixed contract-related transaction costs in return for the timely delivery of a quality product. In the next section, we incorporate such stylized facts into a theoretical model of a modern agricultural economy.

[7] See Key and Runsten (1999) on information costs, Farina (2002) on irrigation technology, Ashraf, Giné, and Karlan (2009) and Asfaw, Mithöfer, and Waibel (2010) on certification, and Blandon, Henson, and Cranfield (2009) on producer cooperatives.

[8] Other benefits include reduced risk and variability as modern-sector output prices can be considerably less volatile (Michelson, Reardon, and Perez, 2011), and resource provision as agro-industrial firms commonly offer specialized inputs (e.g., seeds, fertilizers, pesticides, etc.) to overcome issues associated with thin or missing markets (Key and Runsten, 1999).

3 A New Model of Agrarian Production

Recall that EK predict that egalitarian land distributions enhance productivity and welfare. However, also recall that the EK model lacks a modernized value chain, raising questions about its contemporary policy relevance. In this section, we introduce our new model of agrarian production, which seeks to remedy this shortcoming. The model has two types of agents: a single monopsonistic procurer, and N producers. Section 3.1 provides a detailed discussion of procurer behavior, section 3.2 describes producer behavior, and section 3.3 outlines our computational implementation and presents our baseline parameterization of the model.

3.1 Procurer Behavior

The procurer is a modern, profit-maximizing processor or distributor of agricultural commodities. While the procurer is a price taker in world markets, it has monopsony power locally where it purchases output from contracted growers. We consider a stylized version of the procurer's optimization problem where real procurer profits are

$$\Pi(\rho) = (P - \rho) \cdot Q_M(\rho) \tag{1}$$

and where P is the relative world price (i.e., the procurer's selling price), ρ is the relative procurement price, and Q_M is the contracted quantity for the modern sector.[9] So $(P - \rho) \cdot Q_M$ is real net revenue generated by procurement and distribution.[10]

Prices are relative to the output price in traditional markets. If $\rho \leq 1$ there is no output price premium, so all producers will forego any contractual farming arrangements and produce for the traditional sector. If $\rho \geq P$ the procurer does not find it profitable to contract any producers. Procurer profits are therefore zero for $\rho \leq 1$ or $\rho \geq P$. With ρ as its choice variable, the procurer seeks a constellation of contracted growers that maximizes profits. A profit maximum will be characterized by $\rho \in [1, P]$.

Since producers are attracted to the modern sector by the price premium, Q_M depends on ρ. Suppose for a moment that $Q_M(\rho)$ is increasing at $\rho = 1$, concave, and differentiable. Then the procurer's problem has a predictable solution, with first-order condition

$$0 = (P - \rho) \cdot Q'_M(\rho) - Q_M(\rho) \tag{2}$$

This says that, at the profit maximizing ρ, the profit from marginal procurement must just offset the cost

[9] Letting the procurer also incur contract-related transaction costs does not alter our core results.

[10] We assume that ρ is uniform across contracted producers. Our assumption of a uniform procurement price appears reasonable, since procuring firms conventionally pay uniform prices (Sivramkrishna and Jyotishi, 2008). To illustrate, although Campbell's briefly offered seven different types of contracts in Mexico in the mid-1980s, the array of different procurement prices was short-lived. Implementation was costly and the firm was pressured by other plants to adopt constant and uniform pricing (Key and Runsten, 1999).

of paying all growers a higher price.

However, there is no such simple analytic solution to the procurer problem. One issue is that the response of Q_M to ρ is mediated through factor market adjustments: when the procurer sets ρ, producer incentives change, and this causes adjustments in factor markets that affect supply to the procurer. An additional problem arises as producers decide discretely whether to participate or not. Given a finite set of producers, small changes in ρ can cause discrete changes in supply. We approach these issues computationally. We compute a general equilibrium where the producer's choice of ρ is optimal for the prevailing factor prices, which in turn are an equilibrium response to ρ. (For details, see our online supplementary appendix.)

3.2 Producer Behavior

Producers are price takers and utility maximizers. They are heterogeneous: they differ in the quantity of land owned, and some are landless. Let N_0 and N_1 be the number of landless and land-owning agents, so the total number of producers is $N = N_0 + N_1$. The region has H (abstract) units of land, which are homogeneous in quality, and land ownership is Pareto distributed with equality index $\delta \in (0, 1]$.[11] By increasing δ, we can explore the effect of more egalitarian distributions of landholdings on the model outcomes.

Producers derive utility from real income (Y) and time reserved for non-market activities (t_r). For brevity, we will refer to t_r as leisure time. All our functional forms come directly from EK, so we have

$$U(t_r, Y) = D\sqrt{t_r} + Y \tag{3}$$

where D is a positive parameter. Our characterization of producers follows the EK model, with one exception: producers may now choose to produce for the modern sector. Each producer optimizes their utility by choosing among three activities: pure agricultural laborer, traditional-sector producer, or modern-sector producer (i.e., contract farming). A producer's utility will be maximal over these alternatives:

$$U^* = \max\{U^*_{PL}, U^*_{TS}, U^*_{MS}\} \tag{4}$$

where U^*_{PL}, U^*_{TS}, and U^*_{MS} denote maximal utility as a pure laborer, traditional-sector producer, or modern-sector producer. The payoff in each activity is influenced by the producer's land endowment, factor prices,

[11] Since $\delta = (1 - \text{Gini})/(1 + \text{Gini})$, this is equivalent to parameterizing by the Gini coefficient of inequality. Since we have a finite number of producers, we use a discrete analog to the continuous Lorenz curve used by EK. The Lorenz curve associated with the Pareto distribution can be written as $F(p) = 1 - (1 - p)^\delta$ where $0 < \delta \leq 1$ and p is the cumulative proportion of ranked landowners. In the discrete case, let $p_i = i/N_1$ $(i = 1, \ldots, N_1)$ and set the cumulative land share of producers that own no more land than the i-th landowner to $F(p_i) = 1 - (1 - p_i)^\delta$. Let $\bar{h}_i$ be the land endowment of producer i, sorted by size. Since $\bar{h}_i/H = F(p_i) - F(p_{i-1})$ and $p_i - p_{i-1} = 1/N_1$, we have $\bar{h}_i/H = \left(1 - (i-1)/N_1\right)^\delta - \left(1 - i/N_1\right)^\delta$. Distributional equality (among the landed) arises when δ is unity.

and ρ.

Consider first the pure agricultural laborer, who derives income solely from wage labor and rents derived from letting out any land owned. Accordingly, we have $Y = w\,t_w + v\,\bar{h}$ where w is the real wage, t_w is time spent on wage labor, v is the real rental rate for land, and $\bar{h}$ is the amount of land owned. We normalize to unity the total time available, so for the pure agricultural laborer we have $t_r = 1 - t_w$. The pure agricultural laborer can therefore be characterized by the maximand $D\sqrt{1-t_w} + wt_w + v\bar{h}$, subject to a nonnegativity constraint on t_w. When wages are too low to justify wage labor (i.e., $w < D/2$), we have a corner solution at $t_w^* = 0$. Otherwise, this problem has a unique maximizer, $t_w^* = 1 - D^2/(4w^2)$. We can then substitute t_w^* into the maximand to find U_{PL}^*.

The alternative to pure agricultural labor is agricultural production for sale either into the traditional value chain or into the modern value chain. Given that the optimization problems facing the prospective modern-sector and traditional-sector producers are quite similar, we discuss both problems in parallel. The objective of each producer type is to maximize utility subject to time and working-capital constraints.[12] However, recalling section 2.2, modern-sector producers typically incur additional fixed costs in order to receive the output price premium. Normalizing the output price in traditional spot markets to unity, let the modern-sector producers receive the relative output price $\rho > 1$, which embodies the price premium. Further, denote the fixed costs of participating in the modern (traditional) sector by K_M (K_T), where $K_M > K_T$.

Both land and labor are required for agricultural production. Production technology for a single producer is

$$q = A\sqrt{h\,(t_h + L)} \tag{5}$$

where q is producer output, A is a positive productivity parameter, h is operational landholdings, t_h is own labor applied to the operational landholdings, and L is hired labor. Taking into account possible land leasing and labor market participation, real income is

$$Y = pq + w(t_w - L) + v(\bar{h} - h) - K \tag{6}$$

where $(p, K) = (\rho, K_M)$ for a modern-sector producer and $(p, K) = (1, K_T)$ for a traditional-sector producer. Recalling (3), the maximand for each producer type can therefore be written as follows:

$$U(t_r, Y) = D\sqrt{t_r} + pq + w(t_w - L) + v(\bar{h} - h) - K \tag{7}$$

[12] As argued by EK, the constraints are directly influenced by two market "failures" ubiquitous in developing country agriculture: a labor market imperfection deriving from the incentive for hired labor to shirk, which necessitates supervision and thereby influences time allocation; and a credit market imperfection deriving from the requirement of collateral, which means that a producer's access to working capital depends on the quantity of land owned. (See section 2.1.)

with the appropriate (traditional or modern sector) substitutions made for p and K.

Regarding the time constraint, in addition to the three uses of time defined above (t_r, t_h, and t_w), hired labor requires time t_s for supervision. Naturally, all four uses of time must be nonnegative and all uses of a producer's time must sum to the total available (which is normalized to unity):

$$t_r + t_h + t_w + t_s = 1 \tag{8}$$

Supervision time is a function of hired labor: $t_s = s_1 L + s_2 L^2$, where $s_1 > 0$ and $0 < s_2 < 1$.[13] With respect to the working capital constraint, working capital is required to hire labor and rent land. Collateral-based access to working capital ($\bar{B}$) depends linearly on the amount of land that a producer owns: $\bar{B} = \phi + \theta\bar{h}$, where the parameters ϕ and θ are nonnegative. The working capital constraint can then be written as follows:

$$v(h - \bar{h}) + w(L - t_w) \leq \bar{B} - K \tag{9}$$

(with the appropriate substitution made for K depending on the production sector). All working-capital outlays are incurred at the beginning of the production period.

Given the definition of the utility function and constraints, we can represent the producer's constrained optimization problem with the following Lagrangian:[14]

$$\begin{aligned}\mathcal{L}(t_r, t_h, t_w, L, h, \lambda, \mu) =& D\sqrt{t_r} + pA\sqrt{h\,(t_h + L)} + w(t_w - L) + v(\bar{h} - h) - K \\ &+ \lambda\left[\bar{B} - w(L - t_w) - v(h - \bar{h}) - K\right] \\ &+ \mu\left[1 - t_r - t_h - t_w - s_1 L - s_2 L^2\right]\end{aligned} \tag{10}$$

recalling that $(p, K) = (\rho, K_M)$ for a modern-sector producer and $(p, K) = (1, K_T)$ for a traditional-sector producer. EK analyze this optimization problem in detail.[15]

Let $B \equiv \bar{B} + v\bar{h} - K$. EK show that the unique solution to (10) can be parameterized by B. Table 1 presents the EK solution for four possible modes of production, separated by three critical values of B.

[13] Again, all functional forms are from EK. These functional forms embody standard assumptions. For example, the production function is linear homogeneous, strictly concave, and twice-continuously differentiable. Similarly, the supervision function is strictly convex and twice-continuously differentiable. The conventional justification for this strict convexity is that it ensures a finite farm size despite the linear homogeneity of the production function. The restriction $s_1 < 1$ is needed for hired labor to ever be profitable.

[14] For compactness, we suppress the nonnegativity constraints on the components of time use.

[15] A producer cannot have $t_h = 0$ and $L = 0$ since labor is an essential production input. Additionally, since the effective cost of hired labor exceeds the market wage (i.e., it must be supervised), a producer who hires workers will not also provide wage labor (i.e., t_w and L cannot both be positive). However, it is possible that both $t_w > 0$ and $t_h > 0$, since a producer may also provide some wage labor.

EK assume that the capital constraint always binds on cultivators. This is a very strong assumption as it can force producers to use unwanted working capital and can thereby impose suboptimally high levels of production or an inappropriate abandonment of cultivation. While EK needed this assumption for tractability of their simulation approach, our agent-based approach allows us to discard it.

Table 1: EK's Capital Constrained Class Structure

Class	Characteristics	Working Capital (B)
LC (laborer-cultivator)	$t_w > 0$, $t_h > 0$, $L = 0$	$0 \leq B < B_1$
SC (self-cultivator)	$t_w = 0$, $t_h > 0$, $L = 0$	$B_1 \leq B < B_2$
SM (small capitalist)	$t_w = 0$, $t_h > 0$, $L > 0$	$B_2 \leq B < B_3$
LG (large capitalist)	$t_w = 0$, $t_h = 0$, $L > 0$	$B \geq B_3$

Being a pure laborer is always a reserve option, if cultivating is not preferable. As in EK, agents in our model choose traditional-sector production over pure laboring if $U^*_{TS} > U^*_{PL}$. Given that we also include a modern sector, however, our agents will seek contractual farming arrangements if $U^*_{MS} > \max\{U^*_{PL}, U^*_{TS}\}$. If a producer wishes to participate in the modern sector but the procurer does not offer a contract, the producer will choose the next best alternative.[16]

3.3 Implementation and Baseline Parameterization

We implement our model as an agent-based computational model.[17] Agent-based methods allow very general representations of agent heterogeneity (Epstein and Axtell, 1996), and in our model producer-level variation in landholdings is fundamental. An additional payoff to agent-based methods is that each agent can autonomously optimize based on its particular state, constraints, and goals. In our model, producers have the same goals (i.e., utility functions), but they face substantial constraint heterogeneity since land is used as collateral. It is not obvious how one could correctly handle these constraints without agent-based methods.[18]

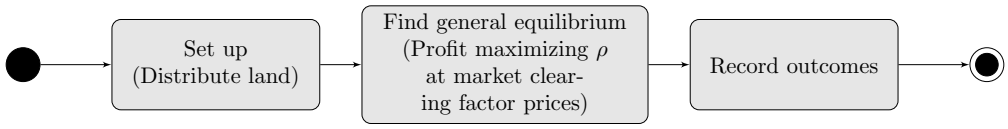

Figure 1: Agent-Based Model Structure

Figure 1 presents an activity diagram of the steps followed to generate the model outcomes. This algorithm is executed once for each considered value of δ, from the least to the most equal distribution of land.[19] The core component of the set-up phase is the distribution of land to producers, based on δ. Land

[16] This will not happen in our baseline parameterization, but it can happen if the procurer incurs contract-related transaction costs from producer participation.

[17] There are many platforms available for agent-based modeling. Our implementation is in Python, a general-purpose, object-oriented programming language. This choice is incidental to our research program, but Python has found wide use in scientific programming (Langtangen, 2012). Our implementation depends on the `gridworld` module described by Isaac (2011), the `numpy` numerical library for Python described by van der Walt, Colbert, and Varoquaux (2011), and the SciPy scientific programming library for Python. See the source code in our supplementary online appendix.

[18] For example, EK achieve tractability in their simulations by assuming the working-capital constraint always binds, but it does not.

[19] In the five-fold categorization of developing countries of Lipton (2009), the average Gini for the 19 countries in his most unequal groups (groups I and II) can be computed as 0.82, using his data. Using the relationship that $\delta = (1 - \text{Gini})/(1 + \text{Gini})$, this is equivalent to a δ value of approximately 0.10. We therefore select this as our initial (most unequal) δ value. (The Gini

ownership influences the factor demands and supplies of producers. Accordingly, we then find the general equilibrium for the given δ.[20] In a general equilibrium, the relative procurement price (ρ) must maximize the monopsonistic procurer's profits at the current factor prices, while those factor prices must be the market clearing prices when growers face that procurement price.[21] Finally, we record the model outcomes associated with the general equilibrium.

We track a variety of aggregate outcomes for producers, including welfare, output, the poverty rate, the proportion in each activity, the proportion in each agricultural class, and the land use of each agricultural class. The reported utility and output aggregates are means across producers. (This allows ready comparisons with scenarios involving variations in the number of producers.) The poverty rate is constructed as the proportion of producers with income below the EK poverty line. We also track outcomes for the procurer, including the procurement price, output procured, and total profit.

Baseline parameter values are listed in Table 2, and most come directly from EK. One new parameter is required by our use of agent-based methods: the number of producers (N). EK distributed producers along a continuum, while in an agent-based implementation we must specify the population size. Two new parameters are introduced when we add a modern value chain: the fixed costs of participation in the modern sector (K_M), and the procurer's sales price (P). Table 2 additionally includes a few derived values, which are chosen to mimic EK to the extent possible. The number of landless is determined by p_0, which corresponds to the proportion landless in EK (see their Figure 2). Total arable land is similarly chosen to match their relationship between landless producers, landholders, and the quantity of arable land.

EK did not calibrate their model, and it would be counterproductive for our project to attempt a full calibration. Our goal is to examine whether adding a modern value chain alters the core predictions of the EK model. Accordingly, we must adhere closely to the original model, so when possible our parameters are taken directly from EK. Of course, this is not possible with our modern sector parameters. Our baseline values for these parameters are roughly informed by empirical studies. For example, Michelson (2013) conducted a supermarket supplier census in Nicaragua and found 244 current suppliers. We set $N = 250$ in the baseline. To cite another example, Schipmann and Qaim (2011) compared product prices across retail outlets in Thailand and found that modern retailer prices ranged from 40 to 340 percent higher than

coefficients used in the calculation correspond to Lipton's most recent data, which spans the years 1990-2005.)

[20] Since our model implementation is agent-based, we are able to use the exact solution for each agent, including careful attention to the constraints facing each agent and the possibility of corner solutions. Each agent computes its unconstrained optimum. If its constraints bind, it computes a constrained optimum. We code analytical solutions to (10). The analytical solution for the working-capital-constrained large capitalist proves surprisingly complicated. See the source code in our supplementary online appendix.

[21] For any given ρ, factor-market clearing is achieved when factor prices (v, w) produce zero excess demand in the land-rental and wage-labor markets. As usual, we find this by numerical search. We use the `fsolve` function from SciPy's `optimize` module. SciPy is a widely-used scientific programming library for Python; `fsolve` is just a wrapper around MINPACK's hybrd and hybrj algorithms. Since $\rho \in (1, P)$, we use a bracketing-interval solver to find the procurer's profit-maximizing ρ (with underlying factor-market clearing). Source code is in our online supplementary appendix.

Table 2: Baseline Parameterization

Parameter	Description	Value
	Parameters from EK	
A	production function productivity ($A\sqrt{h\,(t_h + L)}$)	5.0
D	sub-utility of leisure ($D\,\sqrt{t_r}$)	0.1
s_1, s_2	labor-supervision costs ($s_1 L + s_2 L^2$)	0.3, 0.01
ϕ, θ	land-based working capital ($\phi + \theta\bar{h}$)	0.0, 1.0
p_0	proportion of agents without land	0.33
povertyline	poverty line	1.3
K_T	producer's fixed cost (traditional sector)	0.50
	New Parameters	
N	number of producers	250
K_M	producer's fixed costs (modern sector)	0.75
P	sale price of procurer	3.5
	Derived Values	
N_0	number of landless producers	$p_0 \cdot N$
N_1	number of land-owning producers	$N - N_0$
H	total arable land	$N_1/2 + N_0$

traditional wet markets. We set $P = 3.5$ in our baseline parameterization.[22] While our choices are intended to be reasonable, we will show that our results are robust to deviations from our particular parameterization by supplementing our baseline results with a sensitivity analysis.[23]

4 Results and Sensitivity Analysis

This section comprises two subsections. The first subsection discusses the results associated with our baseline parameterization. The second subsection examines the sensitivity of our core results to changes in the baseline parameters.

4.1 Baseline Results

In the absence of a modern value chain, our baseline model collapses to an agent-based version of the EK model, and our simulation results are therefore very similar to theirs. For example, EK's illustration of land-use patterns (in their Figure 2) can be compared with the first subfigure in our Figure 2, which illustrates producer outcomes in the absence of a modern sector. (When comparing figures, note that the range for δ differs slightly.) Our figure is qualitatively very similar to the EK results.[24] At high levels of inequality

[22] See Minten and Reardon (2008) for further evidence on price differences between traditional and modern retailers. We did not find empirical evidence to inform the baseline for K_M, but it is constrained by two requirements: it must be greater than K_T or we would not see a traditional sector, and it must not be so large that the modern sector disappears.

[23] Effectively we have two baseline parameterizations: one for the traditional economy (the EK model), and one (with three additional parameters) for our modern economy. For compactness usually we refer to both as the baseline parameterization.

[24] Our agent-based results allow a slightly more prominent role for self-cultivators (SC), since our simulation includes a check for whether or not the working capital constraint binds. The EK simulations were not agent-based. Instead, they assumed a continuum of producers and generated simulation results by approximating the implied integrals. For reasons of tractability,

(i.e., low δ), large capitalist (LG) farming dominates—one thinks of South and Central America. But as the distribution of landholdings becomes more equitable, small capitalist enterprises prevail—one thinks of East and Southeast Asia. As is evident from our second subfigure in Figure 2, this pattern of land use is quite robust to the introduction of the modern sector.

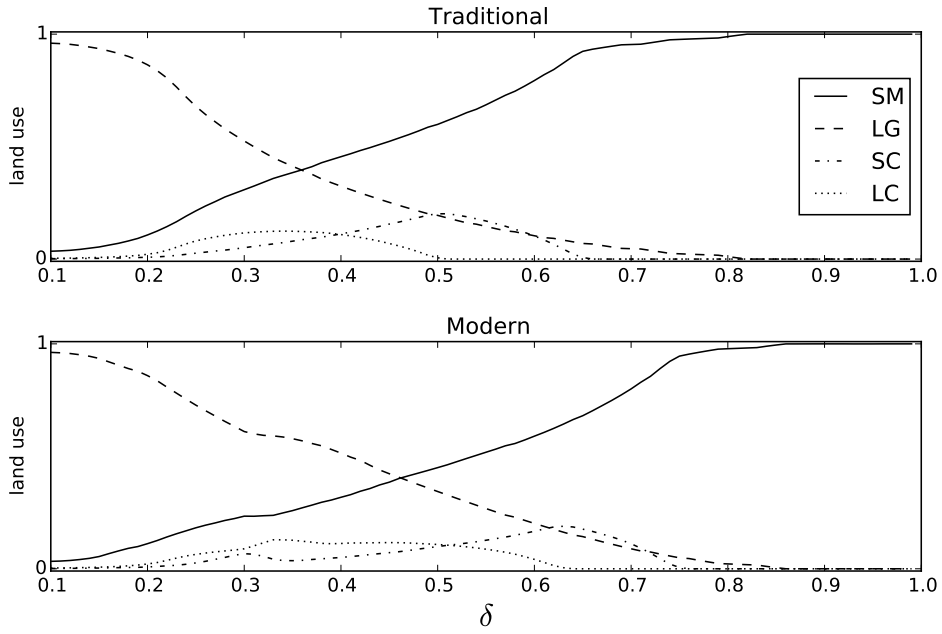

Figure 2: Agrarian Land Use (Traditional vs. Modern)
Land use, which is the proportion of land operated by each class, transitions from being dominated by large capitalists to being dominated by small capitalists as the distributional parameter δ increases. The traditional economy has no modern sector. Depicted classes: laborer-cultivator (LC), self-cultivator (SC), small capitalist (SM), and large capitalist (LG). Parametrization: baseline.

Figure 3 offers a different perspective on the effect of land redistribution on the class structure of the resulting economy. (We find this result to be insensitive to the absence or presence of the modern sector, so we only illustrate the latter case.) The figure is a stacked-bar chart, the height of which is the total population of producers, so for a given δ the relative height of the colored bars illustrates the proportion of producers in each class. We use darker colors for less capital-constrained production activities. White is therefore the pure agricultural laborer (PL), light gray the laborer-cultivator (LC), gray the self-cultivator (SC), dark gray the small capitalist (SM), and black the large capitalist (LG). Clearly the distribution of land has large effects on the class structure of production. As the distribution becomes more egalitarian, the producer population transitions from primarily laboring to primarily cultivating, and small capitalists come to dominate production. Comparing Figures 2 and 3, we see that even when large capitalists dominate land

the EK simulations imposed continual binding of the working capital constraint. This can force overuse of working capital. In our agent-based implementation, we allow each individual producer to determine whether or not the working capital constraint is binding.

use, they are a small proportion of all cultivators. In contrast, when small capitalists dominate land use, they constitute the bulk of all cultivators.

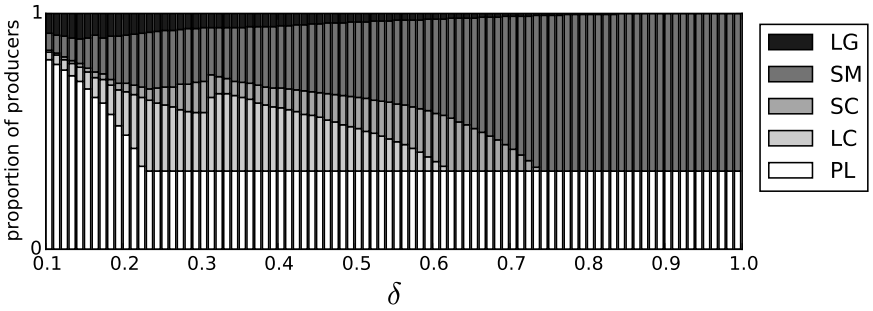

Figure 3: Class Structure and Land Distribution
As δ increases, producers transition from primarily laboring to primarily cultivating, and small capitalists come to dominate production. Depicted classes: pure agricultural laborer (PL), laborer-cultivator (LC), self-cultivator (SC), small capitalist (SM), and large capitalist (LG). Parameterization: baseline.

Despite the small impact of the modern sector on land-use patterns and the class composition of agrarian production, modern-sector participation is substantial. We document this in Figure 4. This figure is another stacked-bar chart, displaying the proportion of producers in each sector for each level of δ. We use white bars for pure agricultural laborers, gray for cultivators that remain in the traditional sector, and dark gray for cultivators that produce for the modern sector. As the land distribution grows more equal, more and more producers can surmount the fixed cost of modern-sector participation, and the modern sector slowly replaces the traditional sector.

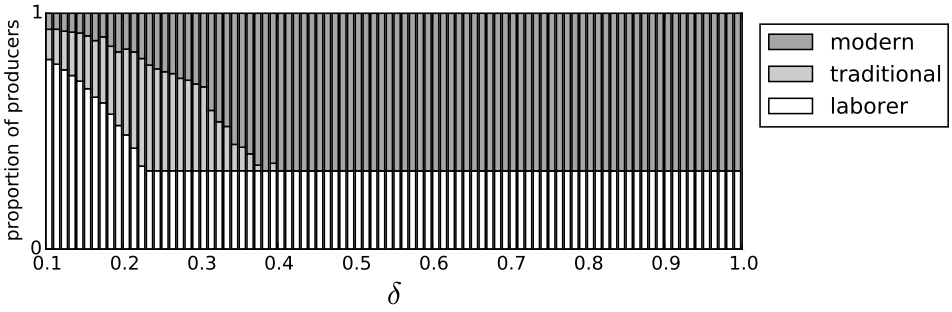

Figure 4: Distribution of Producers Across Sectors
Increases in distributional equality (δ) are associated with fewer pure laborers and more modern sector producers. Traditional sector production vanishes beyond intermediate values of δ. Parameterization: baseline.

EK emphasize that equilibrium in their model involves a misallocation of resources. While the optimal land-to-labor ratios are constant for cultivators with $B < B_1$, they are strictly increasing for $B \geq B_1$.[25]

[25]See Table 1. This is not true when the capital constraint does not bind, but EK assume that it always binds.

Since cultivators set the marginal rate of substitution of land for labor equal to the relative effective factor costs, increases in B (beyond B_1) induce a bias toward land use. This is because the effective cost of labor increases as producers optimally consume less leisure (which raises the price of own labor) and hire more wage labor (which raises the marginal supervision cost). Average land productivity therefore decreases in B. The EK model thereby offers an explanation of the inverse farm-size/productivity relationship, discussed above. For the same reasons, the model implies that land redistribution can raise total agricultural output and improve average producer welfare.

Figure 5 illustrates our results for producer output, welfare, and poverty. The first subfigure illustrates the outcomes for the traditional model, which has no modern sector. The second subfigure illustrates the outcomes for our new model, which includes a modern sector. (In both cases, we use the baseline parameter values, discarding the modern-sector parameters when simulating the traditional model.) The presence of the modern sector has barely detectable effects on output and poverty, but it has substantial effects on average producer welfare. While the introduction of the modern sector increases producer welfare for every value of δ, the relationship between welfare and δ is distinctly non-monotonic. This relationship is of particular interest as a counterpoint to EK, who find that "an increase in the distributional parameter δ ... causes an increase in social welfare ... a direct consequence of the inverse relationship between farm size and land productivity" (Eswaran and Kotwal, 1986, p. 494).

Intuitively, the welfare premium associated with the introduction of the modern sector naturally increases as participation in that sector increases. Referencing Figure 4, we see that participation peaks at $\delta \approx 0.4$, about the same as the welfare peak. For a given producer to incur the cost of participating in the modern sector, output must be sufficiently large for the modern sector premium to offset those costs. High levels of inequality (i.e., low values of δ) are associated with the existence of few farms large enough to justify participation in the modern sector. Interestingly, however, when land inequality falls further (i.e., at higher values of δ) producer welfare declines. This is because, when many producers have shifted to the modern sector at higher levels of land equality, the procurer can use a lower ρ to draw production to the modern sector. Below we show that ρ is indeed decreasing in δ for $\delta > 0.4$, and that this result is robust to large parameter variations. From a policy perspective, this suggests that land redistribution can become counterproductive beyond a certain point.

Finally, Figure 6 illustrates the baseline procurer outcomes. It is evident that the optimal procurement price (ρ) peaks at $\delta \approx 0.4$. As mentioned above, this coincides with peak participation in the modern sector, after which point the procurer uses a lower procurement price to draw production to the modern sector. Even when rising δ drives ρ down, it continues to drive up procurer purchases (Q_M) and profit (Π), until they eventually level off. The monotonic relationship between Π and δ contrasts starkly with the non-monotonic

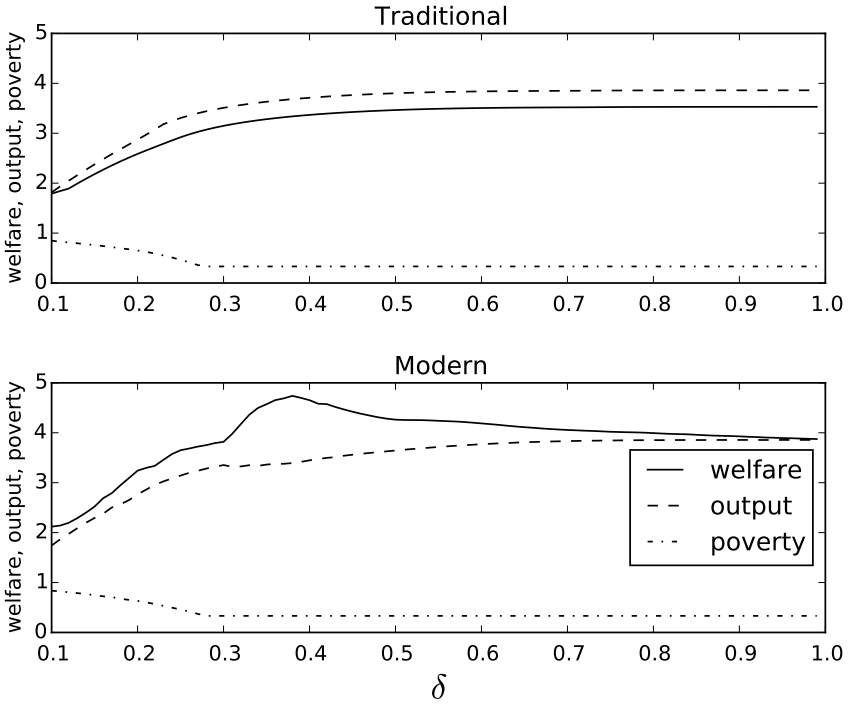

Figure 5: Producer Outcomes (Traditional vs. Modern)
A non-monotonic relationship between the distributional parameter δ and producer welfare arises with the addition of the modern sector. Parameterization: baseline.

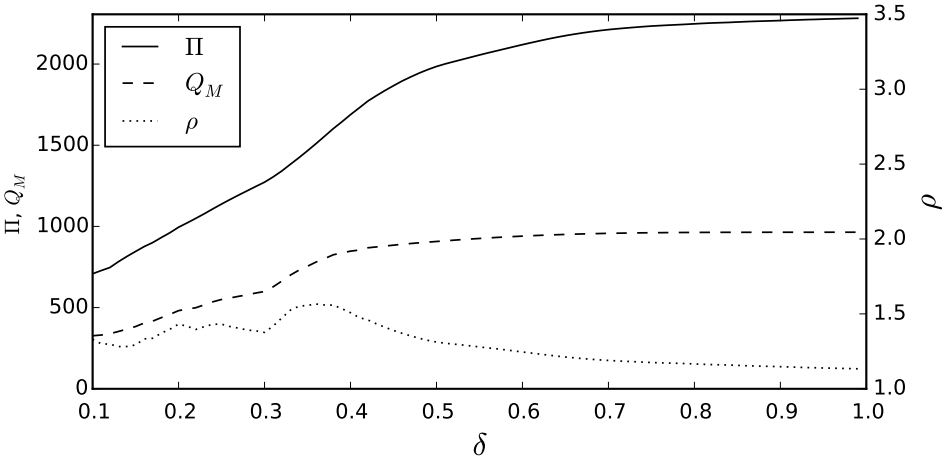

Figure 6: Aggregate Procurer Outcomes
Variations in distributional equality (δ) affect the procurer's optimal procurement price (ρ) and thereby procurer purchases (Q_M) and profits (Π). Purchases and profits are increasing in δ, especially at low δ.

equity-welfare relationship associated with the producer outcomes. These results are robust to changes in the parameter values. In particular, we find qualitatively similar results for large changes in the procurer's output price (P). (See the online supplementary appendix for details.)

4.2 Sensitivity Analysis

We have found that the core prediction of the EK model is overturned when adding a modern sector to the model. To this point, however, we have shown that only for the baseline parameterization. While it is interesting to find a specific parameter set that overturns their prediction, a robust result would be much more interesting. We therefore proceed to sensitivity analysis. (See our online supplementary appendix for additional detail.) Recall that our new model of agrarian production requires three new parameters: N, P, and K_M. Also recall that our goal is to show that—on its own—the emergence of a modern sector undermines EK's core prediction. While we must then use EK's parameters to the extent possible, a sensitivity analysis for the new parameters is desirable.

First, consider the number of producers (N). The EK model worked with a continuum of agents, but our agent-based implementation must specify the number of agents. However, we scale the available land so that land per producer matches the EK specification. (See Table 2.) This ensures that our results are comparable over a wide range of values for N. We consider large variations in the number of producers (i.e., halving or doubling). Here we simply state that, as expected, these variations have negligible effects. (See our supplementary online appendix for more details.)

Next, consider the procurer's sales price (P). Halving and doubling this price relative to the baseline naturally has large effects on procurer profitability. A higher external price also encourages the procurer to bid for more output, so the sales price can have important effects on producer welfare. This is illustrated in Figure 7. At the low sales price depicted in the first subfigure, we can barely perceive the non-monotonic equity-welfare relationship, but it remains. At the high sales price depicted in the second subfigure, it is prominent. Since the procurer can offer little price premium when the sales price is low, these results are also expected.

Finally, we consider the cost of modern-sector participation (K_M). Recalling (6), we expect to observe that K_M is a key determinant of the desirability of modern-sector participation. One way to illustrate this sensitivity is offered in Figure 8, which varies K_M from a low of 0.60 to a high of 1.50. As expected, an increase in K_M produces an decrease in modern-sector participation. With high enough K_M, one can drive out the modern sector altogether. Conversely, as K_M falls toward K_T, eventually all cultivators will prefer to produce for the modern sector.

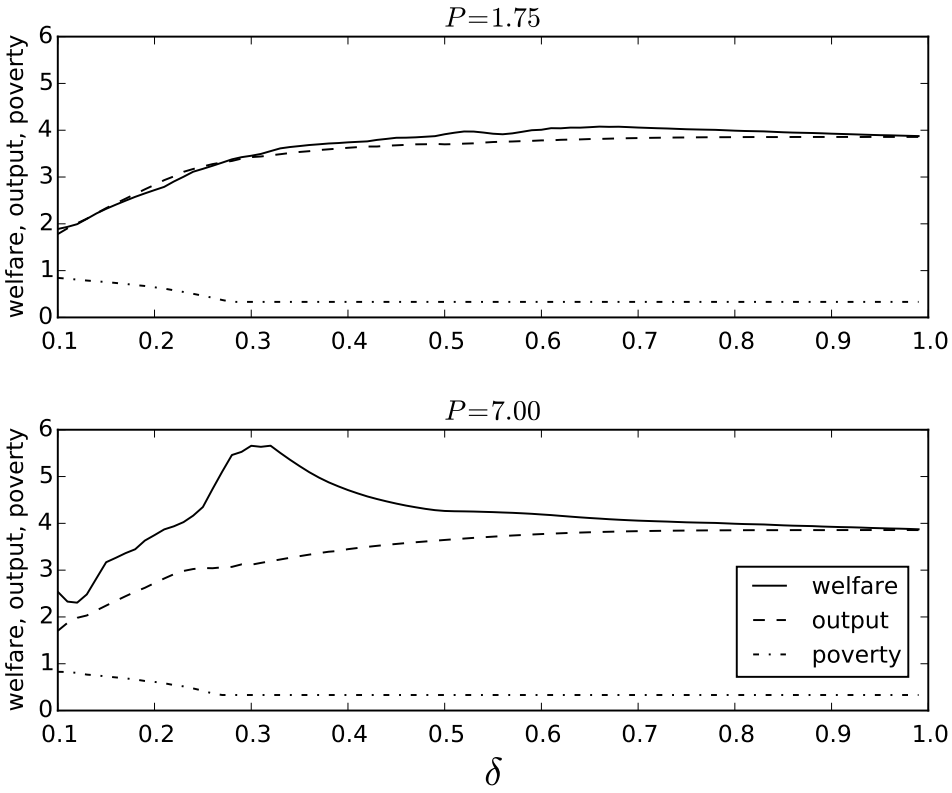

Figure 7: Welfare Effects of Changing P

At a low procurer sales price (P), the non-monotonic equity-welfare relationship remains, but is barely detectable. At a high procurer sales price, the relationship is more pronounced.

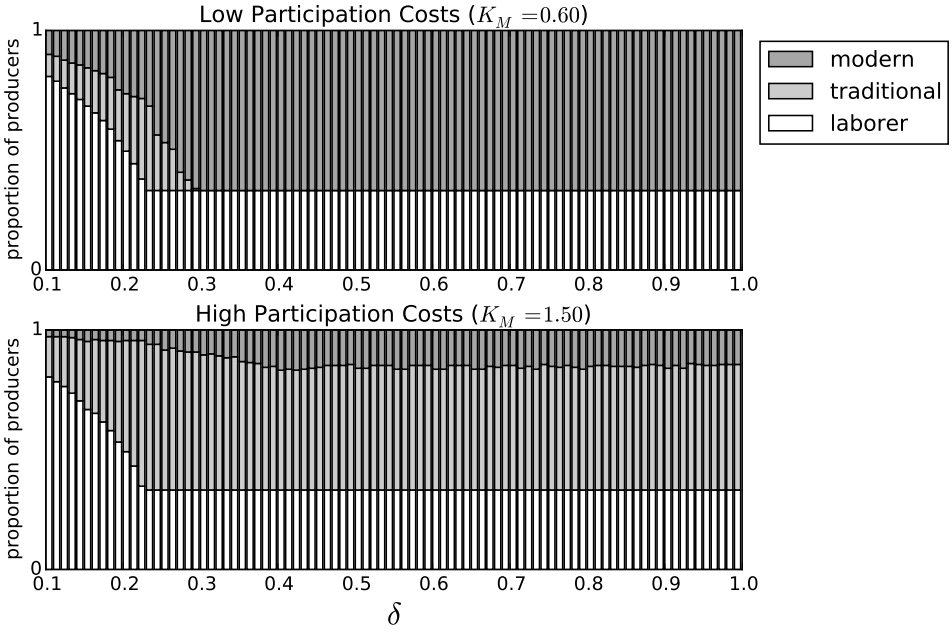

Figure 8: Producer Sectors (Low vs. High Participation Costs)
Higher participation costs (K_M) diminish modern-sector participation, especially when the land distribution is very unequal (low δ). The proportion of pure laborers is unaffected by participation costs.

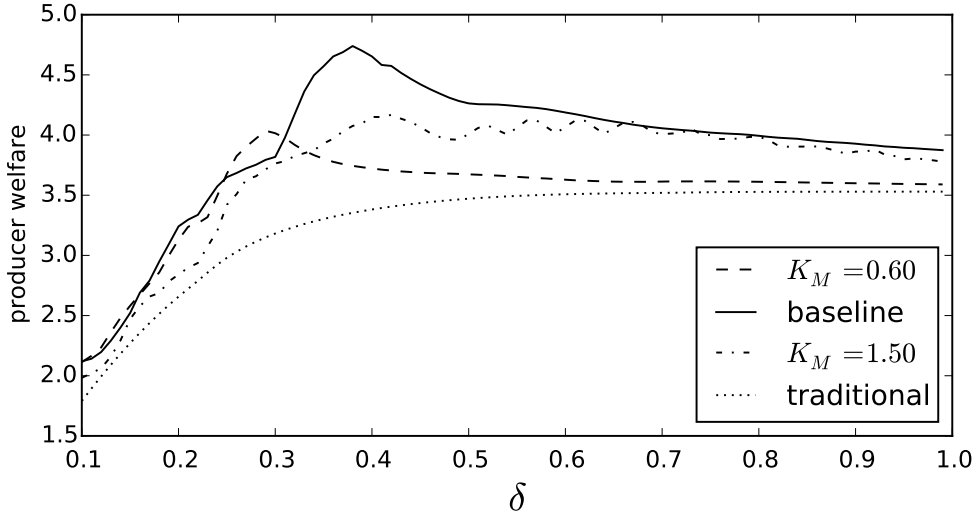

Figure 9: K_M and Producer Welfare
The relationship between modern-sector participation costs (K_M) and mean producer welfare is complex. Even with substantial variation in K_M, however, no monotonic equity-welfare relationship is recovered.

We might expect that higher participation costs will not only lower modern-sector participation, but will also imply lower welfare payoffs to producers. However, the situation is more complicated: a lower participation cost is unambiguously better for any single producer, ceteris paribus, but participation costs have general equilibrium effects. Figure 8 showed us that participation in the modern sector falls as K_M rises. However, an increase in K_M works against the monopsony power of the procurer, as higher ρ is required to draw forth the same level of production. This provides a gain to intramarginal producers. As illustrated in Figure 9, the gain may be large enough to raise average producer welfare.

5 Summary and Conclusion

Empirical evidence suggests that smaller-scale producers have historically possessed a productivity advantage in labor-abundant agrarian economies. In this context, researchers have argued that redistributive land reform may not only improve agricultural productivity, but also increase equity and reduce poverty. Dietary diversification, foreign direct investment, and changing technology have, however, recently induced fundamental changes in agricultural value chains in developing countries. Furthermore, the associated development of modern procurement systems has led to the imposition of quality standards that credit-constrained, small-scale producers typically find difficult to meet. This radical restructuring of global agri-food systems raises questions as to whether redistributive measures continue to possess welfare-enhancing potential.

In recognition of the persistent growth of high-value markets, we re-examine the production and welfare consequences of the distribution of agricultural landholdings. To this end, we extend the influential theoretical work of Eswaran and Kotwal (1986). The EK model famously predicts that a more egalitarian distribution of landholdings will generate increases in aggregate output and welfare. However, the model has a substantive shortcoming: the absence of a modernized agricultural value chain. Our new model of agrarian production includes a modern sector and re-examines the implications of more egalitarian distributions of agricultural landholdings. The inverse farm-size/productivity relationship persists in our model, but EK's core prediction is undermined. In our model, the response of producer welfare to more egalitarian agricultural landholdings is non-monotonic. Past a certain point, a more egalitarian distribution of landholdings diminishes producer welfare. Contrary to the famous EK result, we predict that redistributive measures can indeed become counterproductive.

Our result is the product of scale-biased participation in modern value chains. For a given producer to incur the additional costs of participating in the modern sector, output must be sufficiently large for the associated price premium to offset those costs. High levels of inequality are therefore associated with the existence of few producers large enough to justify participation in the modern sector. When the land

distribution is more egalitarian, modern-sector participation is then naturally higher. Interestingly, with more modern-sector participants, the procurer's profit-maximizing procurement price can decline, lowering the welfare of intramarginal producers. The welfare gain from modern-sector production thus peaks at intermediate levels of land equality, producing a non-monotonic equity-welfare relationship.

A single contradictory scenario is enough to raise questions about the appropriateness of relying on the EK prediction in policy discussions, but we provide more than that. Our sensitivity analysis shows that our result is quite robust. Accordingly, we claim to have overturned the EK prediction.

We believe our new model of agrarian production advances the understanding of developing agrarian economies. Nevertheless, it has important limitations: it is a static general-equilibrium model, and correspondingly it assumes perfect contractual compliance. Barrett et al. (2012) suggest numerous considerations for a more detailed treatment of contracting between procurers and producers. Informational asymmetries between contracting parties, and costly contractual enforcement create space for breach of contract. A procurer may delay or default on the final payment, inappropriately reject product, or lower the procurement price post-harvest. Producers may refrain from adhering to the agreed upon production schedule, engage in side-selling, or fail to make timely product deliveries of sufficient quantity and quality. Contractual breakdown has clear short-run consequences (e.g., payment default). Additionally, there are many long-run consequences (e.g., delisting underperforming producers). Handling these appropriately may require the development of an apt repeated game. Such considerations may further qualify our results, and we hope to explore them in future research.

References

Alvarado, I., and K. Charmel. 2002. "The Rapid Rise of Supermarkets in Costa Rica: Impact on Horticultural Markets." *Development Policy Review* 20:473–485.

Asfaw, S., D. Mithöfer, and H. Waibel. 2010. "What Impact Are EU Supermarket Standards Having on Developing Countries' Export of High-Value Horticultural Products? Evidence from Kenya." *Journal of International Food & Agribusiness Marketing* 22:252–276.

Ashraf, N., X. Giné, and D. Karlan. 2009. "Finding Missing Markets (and a Disturbing Epilogue): Evidence from an Export Crop Adoption and Marketing Intervention in Kenya." *American Journal of Agricultural Economics* 91:973–990.

Assunção, J., and L.H. Braido. 2007. "Testing Household-Specific Explanations for the Inverse Productivity Relationship." *American Journal of Agricultural Economics* 89:980–990.

Barraclough, S. 1973. *Agrarian Structure in Latin America: A Resume of the CIDA Land Tenure Studies of Argentina, Brazil, Chile, Colombia, Ecuador, Guatemala, Peru*. Lexington, MA: Lexington Books, Edited by Solon Barraclough, in collaboration with Juan Carlos Collarte.

Barrett, C., M. Bellemare, and J. Hou. 2010. "Reconsidering Conventional Explanations of the Inverse Productivity-Size Relationship." *World Development* 38:88–97.

Barrett, C.B., M.E. Bachke, M.F. Bellemare, H.C. Michelson, S. Narayanan, and T.F. Walker. 2012. "Smallholder Participation in Contract Farming: Comparative Evidence from Five Countries." *World Development* 40:715–730.

Berry, R., and W. Cline. 1979. *Agrarian Structure and Productivity in Developing Countries*. Baltimore: Johns Hopkins University Press.

Bhagwati, J., and S. Chakravarty. 1969. "Contributions to Indian economic analysis: A survey." *American Economic Review* 59:1–73.

Bhalla, S., and P. Roy. 1988. "Mis-specification in Farm Productivity Analysis: The Role of Land Quality." *Oxford Economic Papers* 40:55–73.

Binswanger, H., K. Deininger, and G. Feder. 1995. "Power, Distortions, Revolt and Reform in Agricultural Land Relations." In J. Behrman and T. Srinivasan, eds. *Handbook of Development Economics*. Amsterdam: North-Holland, vol. 3, pp. 2659–2772.

Blandon, J., S. Henson, and J. Cranfield. 2009. "Small-scale Farmer Participation in New Agri-food Supply Chains: Case of the Supermarket Supply Chain for Fruit and Vegetables in Honduras." *Journal of International Development* 21:971–984.

Bolwig, S., P. Gibbon, and S. Jones. 2009. "The Economics of Smallholder Organic Contract Farming in Tropical Africa." *World Development* 37:1094–1104.

Boselie, D., S. Henson, and D. Weatherspoon. 2003. "Supermarket Procurement Practices in Developing Countries: Redefining the Roles of the Public and Private Sectors." *American Journal of Agricultural Economics* 85:1155–1161.

Carletto, C., S. Savastano, and A. Zezza. 2013. "Fact or Artifact: The Impact of Measurement Errors on the Farm Size-Productivity Relationship." *Journal of Development Economics* 103:254–261.

Carter, M.R. 1988. "Equilibrium Credit Rationing of Small Farm Agriculture." *Journal of Development Economics* 28:83–103.

—. 1984. "Identification of the Inverse Relationship between Farm size and Productivity: An Empirical Analysis of Peasant Agricultural Production." *Oxford Economic Papers* 36:131–145.

Collier, P., and S. Dercon. 2009. "African Agriculture in 50 Years: Smallholders in a Rapidly Changing World?" In *How to Feed the World in 2050: Proceedings of a Technical Meeting of Experts*. Rome, Italy: Food and Agriculture Organization of the United Nations (FAO), pp. 1–13.

Cornia, G. 1985. "Farm Size, Land Yields, and the Agricultural Production Function: An Analysis of Fifteen Developing Countries." *World Development* 13:513–534.

Deininger, K., E. Zegarra, and I. Lavadenz. 2003. "Determinants and Impacts of Rural Land Market Activity: Evidence from Nicaragua." *World Development* 31:1385–1404.

Dries, L., T. Reardon, and J. Swinnen. 2004. "The Rapid Rise of Supermarkets in Central and Eastern Europe: Implications for the Agrifood Sector and Rural Development." *Development Policy Review* 22:525–556.

Eastwood, R., M. Lipton, and A. Newell. 2010. "Farm Size." In R. Evenson and P. Pingali, eds. *Handbook of Agricultural Economics*. Amsterdam: North-Holland, vol. 4, pp. 3323–3397.

Epstein, J., and R. Axtell. 1996. *Growing Artificial Societies: Social Science from the Bottom Up*. The MIT Press.

Escobal, J., and D. Cavero. 2011. "Transaction Costs, Institutional Arrangements and Inequality Outcomes: Potato Marketing by Small Producers in Rural Peru." *World Development* 40:329–341.

Eswaran, M., and A. Kotwal. 1986. "Access to Capital and Agrarian Production Organisation." *The Economic Journal* 96:482–498.

FAO. 2011. *The State of Food Insecurity in the World: How Does International Price Volatility Affect Domestic Economies and Food Security?*. Rome: FAO.

Farina, E.M. 2002. "Consolidation, Multinationalisation, and Competition in Brazil: Impacts on Horticulture and Dairy Products Systems." *Development Policy Review* 20:441–457.

Feder, G. 1985. "The Relation between Farm Size and Farm Productivity: The Role of Family Labor, Supervision and Credit Constraints." *Journal of Development Economics* 18:297–313.

Heltberg, R. 1998. "Rural Market Imperfections and the Farm Size–Productivity Relationship: Evidence from Pakistan." *World Development* 26:1807–1826.

Henderson, H. 2015. "Considering Technical and Allocative Efficiency in the Inverse Farm Size–Productivity Relationship." *Journal of Agricultural Economics* 66:442–469.

Herath, D., and A. Weersink. 2009. "From Plantations to Smallholder Production: The Role of Policy in the Reorganization of the Sri Lankan Tea Sector." *World Development* 37:1759–1772.

Hernández, R., T. Reardon, and J. Berdegué. 2007. "Supermarkets, Wholesalers, and Tomato Growers in Guatemala." *Agricultural Economics* 36:281–290.

Hu, D., T. Reardon, S. Rozelle, P. Timmer, and H. Wang. 2004. "The Emergence of Supermarkets with Chinese Characteristics: Challenges and Opportunities for China's Agricultural Development." *Development Policy Review* 22:557–586.

Isaac, A.G. 2011. "The ABM Template Models: A Reformulation with Reference Implementations." *Journal of Artificial Societies and Social Simulation* 14:(article 5), `http://jasss.soc.surrey.ac.uk/14/2/5.html`.

Keswell, M., and M.R. Carter. 2014. "Poverty and Land Redistribution." *Journal of Development Economics* 110:250–261.

Key, N., and D. Runsten. 1999. "Contract Farming, Smallholders, and Rural Development in Latin America: The Organization of Agroprocessing Firms and the Scale of Outgrower Production." *World Development* 27:381–401.

Langtangen, H.P. 2012. *Python Scripting for Computational Science*, 4th ed. New York: Springer.

Lipton, M. 2009. *Land Reform in Developing Countries: Property Rights and Property Wrongs*. New York: Routledge.

Maertens, M., and J. Swinnen. 2009. "Trade, Standards, and Poverty: Evidence from Senegal." *World Development* 37:161–178.

McCullough, E., P. Pingali, and K. Stamoulis. 2008. "Small Farms and the Transformation of Food Systems: An Overview." In E. McCullough, P. Pingali, and K. Stamoulis, eds. *The Transformation of Agri-food systems: Globalization, Supply Chains and Smallholder Farms*. Rome: FAO, pp. 3–46.

Michelson, H., T. Reardon, and F. Perez. 2011. "Small Farmers and Big Retail: Trade-Offs of Supplying Supermarkets in Nicaragua." *World Development* 40:342–354.

Michelson, H.C. 2013. "Small Farmers, NGOs, and a Walmart World: Welfare Effects of Supermarkets Operating In Nicaragua." *American Journal of Agricultural Economics* 95:628–649.

Minten, B., L. Randrianarison, and J. Swinnen. 2009. "Global Retail Chains and Poor Farmers: Evidence from Madagascar." *World Development* 37:1728–1741.

Minten, B., and T. Reardon. 2008. "Food prices, quality, and quality's pricing in supermarkets versus traditional markets in developing countries." *Applied Economic Perspectives and Policy* 30:480–490.

Miyata, S., N. Minot, and D. Hu. 2009. "Impact of Contract Farming on Income: Linking Small Farmers, Packers, and Supermarkets in China." *World Development* 37:1781–1790.

Neven, D., M. Odera, T. Reardon, and H. Wang. 2009. "Kenyan Supermarkets, Emerging Middle-Class Horticultural Farmers, and Employment Impacts on the Rural Poor." *World Development* 37:1802–1811.

Rao, E., and M. Qaim. 2011. "Supermarkets, Farm Household Income, and Poverty: Insights from Kenya." *World Development* 39:784–796.

Reardon, T., C. Barrett, J. Berdegué, and J. Swinnen. 2009. "Agrifood Industry Transformation and Small Farmers in Developing Countries." *World Development* 37:1717–1727.

Reardon, T., and J. Berdegué. 2002. "The Rapid Rise of Supermarkets in Latin America: Challenges and Opportunities for Development." *Development Policy Review* 20:371–388.

Reardon, T., C. Timmer, C. Barrett, and J. Berdegué. 2003. "The Rise of Supermarkets in Africa, Asia, and Latin America." *American Journal of Agricultural Economics* 85:1140–1146.

Reardon, T., C.P. Timmer, and J. Berdegué. 2008. "The Rapid Rise of Supermarkets in Developing Countries: Induced Organizational, Institutional and Technological Change in Agri-Food Systems." In E. B. McCullough, P. L. Pingali, and K. G. Stamoulis, eds. *The Transformation of Agri-Food Systems: Globalization, Supply Chains and Smallholder Farms*. Rome: FAO, pp. 47–65.

Roemer, J.E. 1982. *A General Theory of Exploitation and Class*. Cambridge, MA: Harvard University Press.

Santos, P., and C. Barrett. 2011. "Persistent Poverty and Informal Credit." *Journal of Development Economics* 96:337–347.

Schipmann, C., and M. Qaim. 2011. "Modern food retailers and traditional markets in developing countries: comparing quality, prices, and competition strategies in Thailand." *Applied Economic Perspectives and Policy* 33:345–362.

Sen, A. 1962. "An Aspect of Indian Agriculture." *Economic Weekly* 14:243–246.

—. 1966. "Peasants and dualism with or without surplus labor." *The Journal of Political Economy* 74:425–450.

Simmons, P., P. Winters, and I. Patrick. 2005. "An Analysis of Contract Farming in East Java, Bali, and Lombok, Indonesia." *Agricultural Economics* 33:513–525.

Singh, S. 2002. "Contracting Out Solutions: Political Economy of Contract Farming in the Indian Punjab." *World Development* 30:1621–1638.

Sivramkrishna, S., and A. Jyotishi. 2008. "Monopsonistic Exploitation in Contract Farming: Articulating a Strategy for Grower Cooperation." *Journal of International Development* 20:280–296.

Stiglitz, J., and A. Weiss. 1981. "Credit Rationing in Markets with Imperfect Information." *American Economic Review* 71:393–410.

Stringer, R., N. Sang, and A. Croppenstedt. 2009. "Producers, Processors, and Procurement Decisions: The Case of Vegetable Supply Chains in China." *World Development* 37:1773–1780.

Trienekens, J., and S. Willems. 2007. "Innovation and Governance in International Food Supply Chains: The Cases of Ghanaian Pineapples and South African Grapes." *International Food and Agribusiness Management Review* 10:42–63.

van der Walt, S., S.C. Colbert, and G. Varoquaux. 2011. "The NumPy Array: A Structure for Efficient Numerical Computation." *Computing in Science & Engineering* 13:22–30.

van Zyl, J., H. Binswanger, and C. Thirtle. 1995. "The Relationship between Farm Size and Efficiency in South African Agriculture." Policy Research Working Paper No. 1548, World Bank, Nov.

Warning, M., and N. Key. 2002. "The Social Performance and Distributional Consequences of Contract Farming: An Equilibrium Analysis of the *Arachide de Bouche* Program in Senegal." *World Development* 30:255–263.

Yotopoulos, P.A., and L.J. Lau. 1973. "A Test for Relative Economic Efficiency: Some Further Results." *American Economic Review* 63:214–223.

What follows is not part of the paper. It will constitute an online supplement to the paper, following the AJAE's *Instructions for Articles with Supplementary Appendix Material* at `http://www.oxfordjournals.org/our_journals/ajae/for_authors/supplementary_appendices.pdf`.

Title: "AJAE appendix for *Modern Value Chains and the Organization of Agrarian Production*"

Date: June 30, 2015

Note: The material contained herein is supplementary to the article named in the title and published in the American Journal of Agricultural Economics (AJAE).

A Supplement to Sensitivity Analyses

This section provides some additional illustrative graphs to supplement the sensitivity analysis in the paper.

A.1 Class Participation

Figure 3 in the paper showed the distribution of sector participation when there is a modern sector. In Figure 10, we additionally show the distribution prior to the introduction of the modern sector (i.e., in the EK model). The qualitative behavior is quite similar.

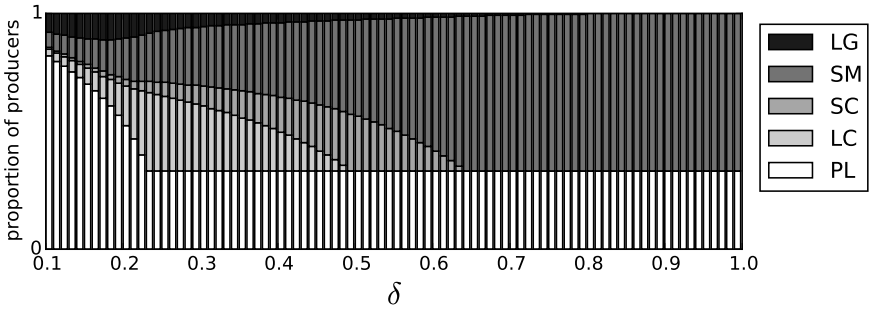

Figure 10: Class Structure and Land Distribution (No Modern Sector)

A.2 Sector Participation

Figure 4 in the paper showed the distribution of sector participation when there is a modern sector. In Figure 11, we additionally show the distribution prior to the introduction of the modern sector. The pattern of choice between cultivating or laboring remains very similar. (Naturally, there are now no modern-sector agents; the legend is retained just to match the original figure.)

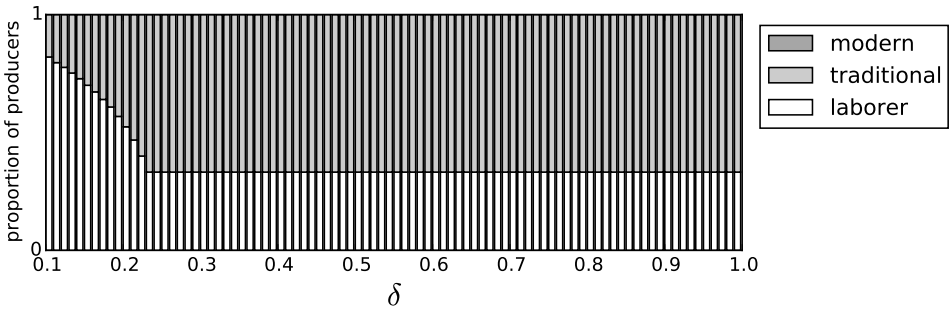

Figure 11: Distribution of Producers Across Sectors (No Modern Sector)

A.3 Sensitivity to Number of Agents

Our core result is Figure 5. The simulation results are very robust to the number of agents. The baseline number of agents is 250. Here we create comparable figures for variations in the number of agents. We always find our core result, the non-monotonic equity-welfare relationship, with a very slight shift to the right of the peak when number of agents increases.

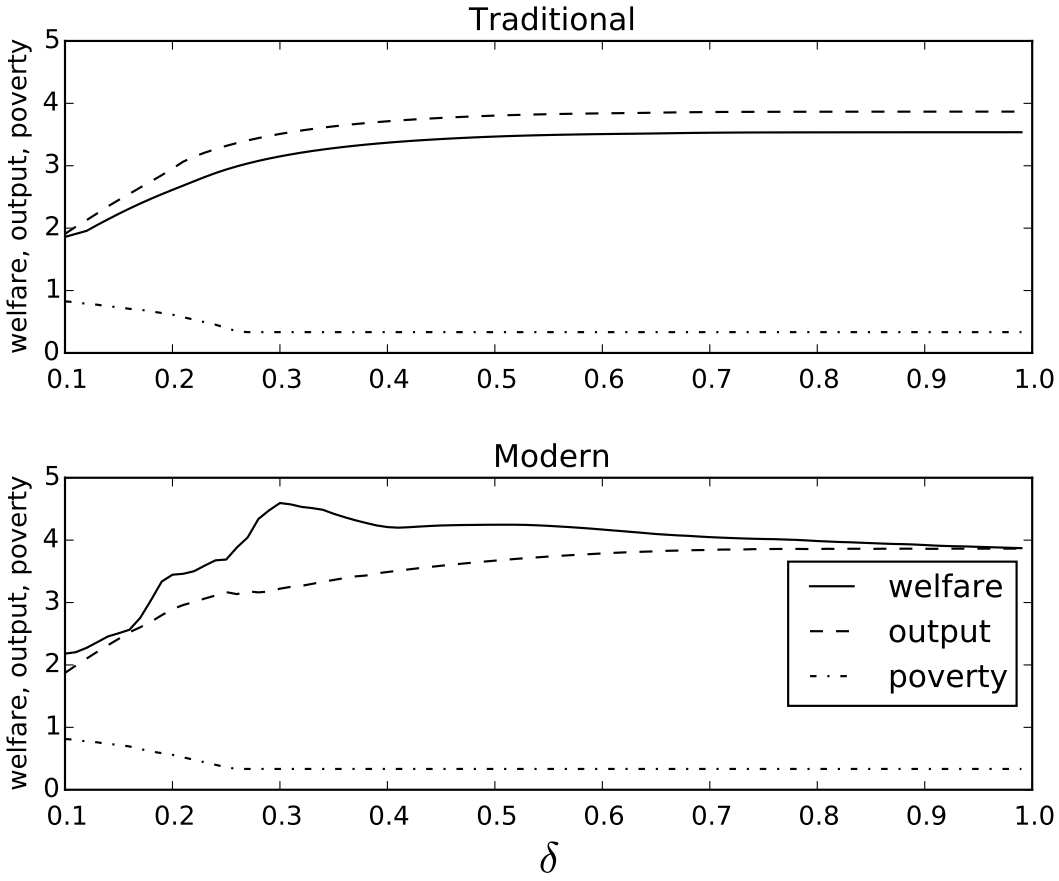

Figure 12: Producer Outcomes with 75 Producers

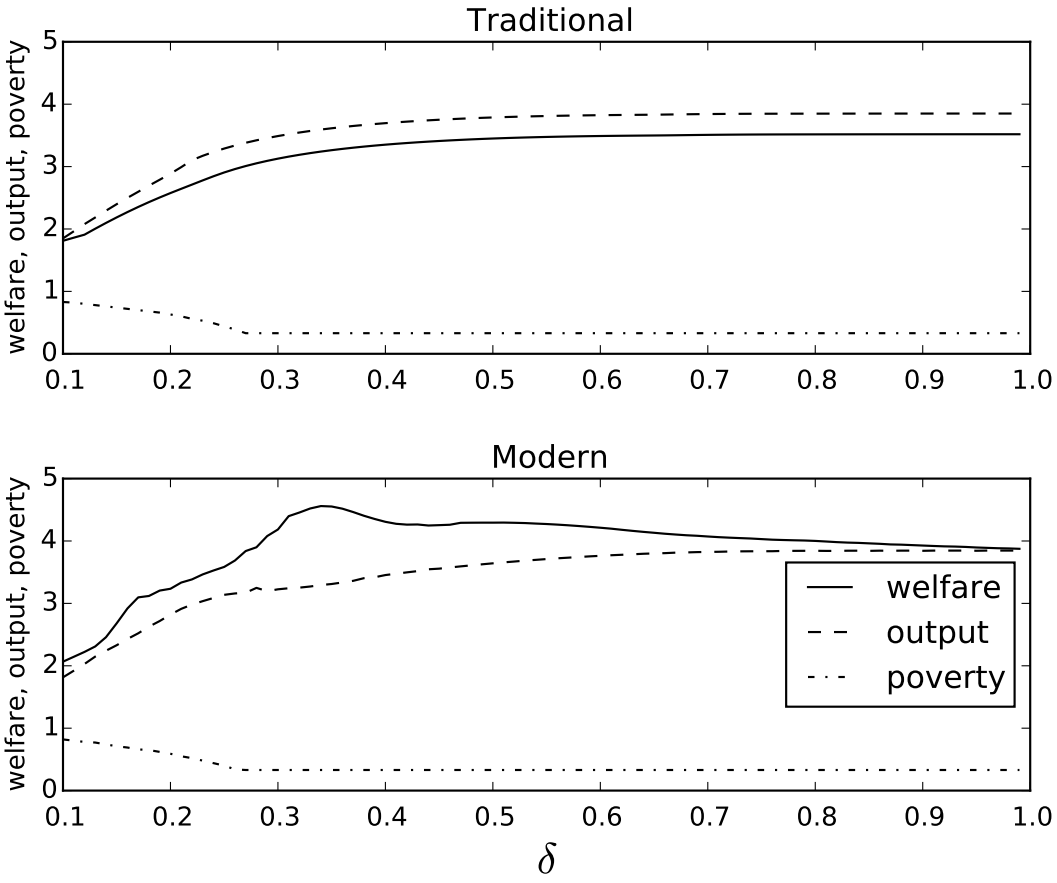

Figure 13: Producer Outcomes with 100 Producers

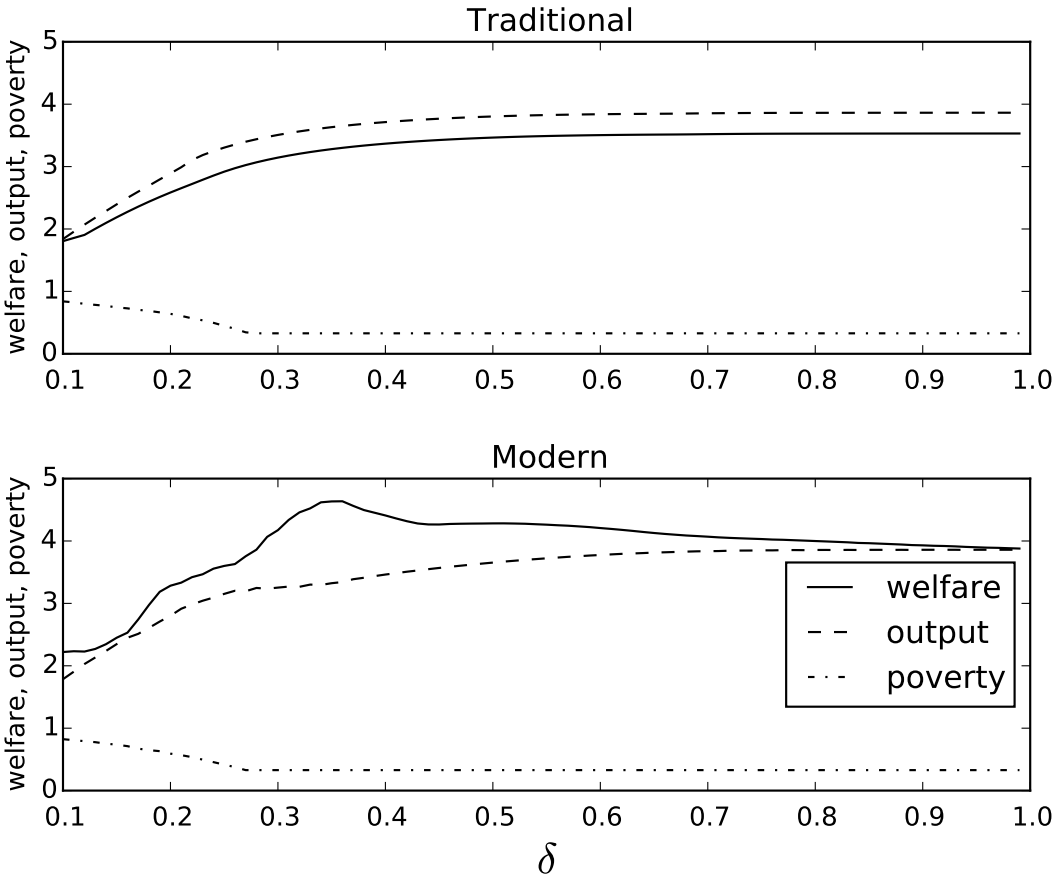

Figure 14: Producer Outcomes with 125 Producers

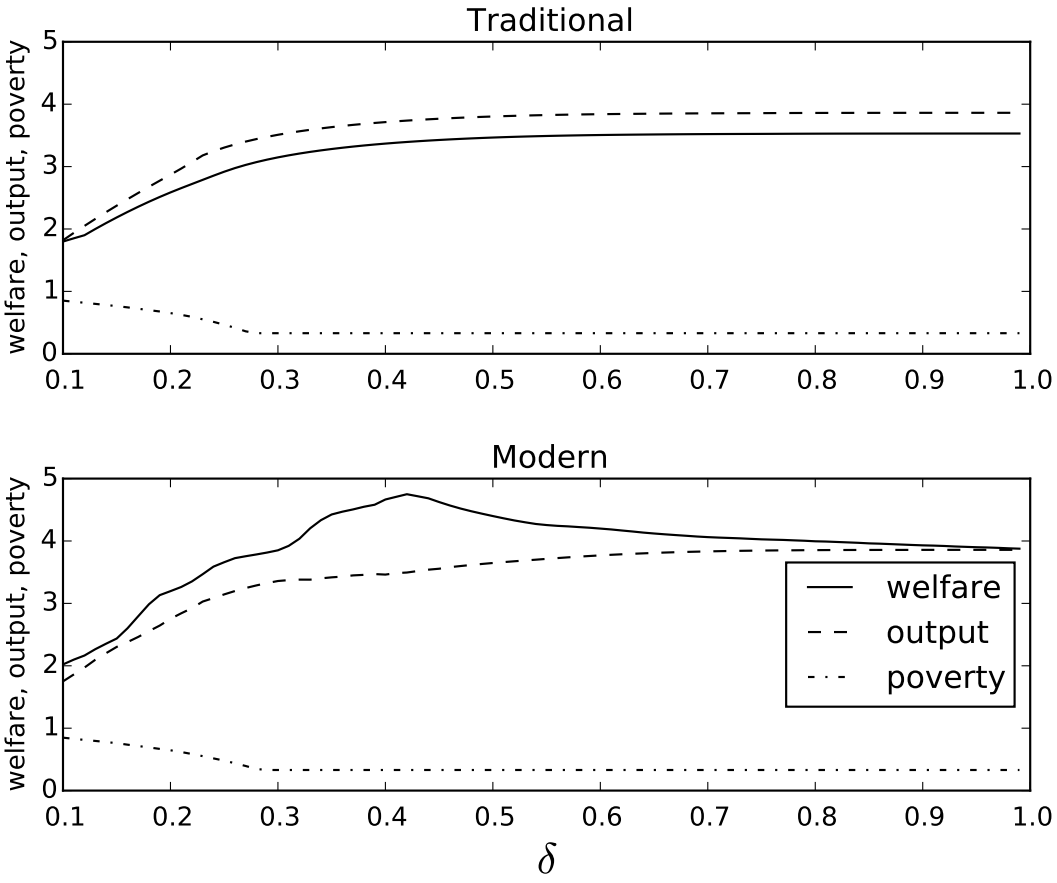

Figure 15: Producer Outcomes with 500 Producers

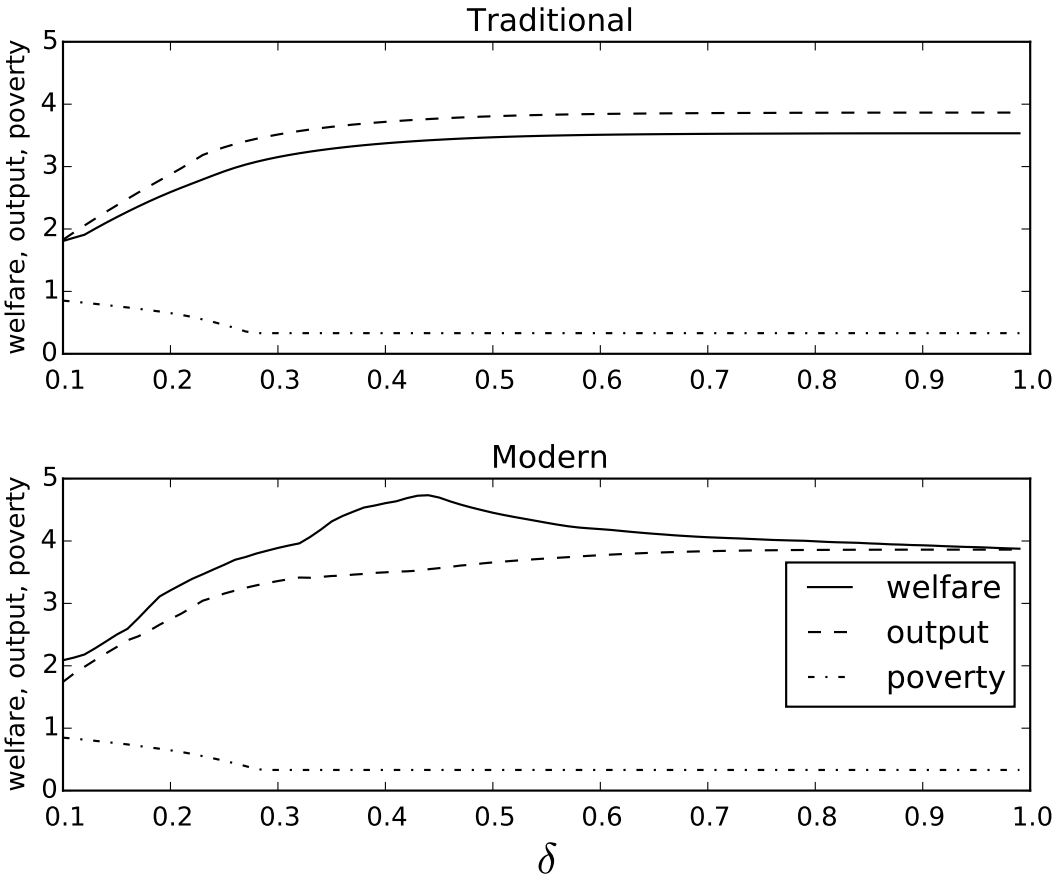

Figure 16: Producer Outcomes with 750 Producers

A.4 Welfare Effects of P

Here we supplement Figure 7 of the paper by looking at additional values of the producer's output price (P). At low external prices, the non-monotonic equity-welfare relationship is barely distinguishable. At higher prices, it manifests clearly.

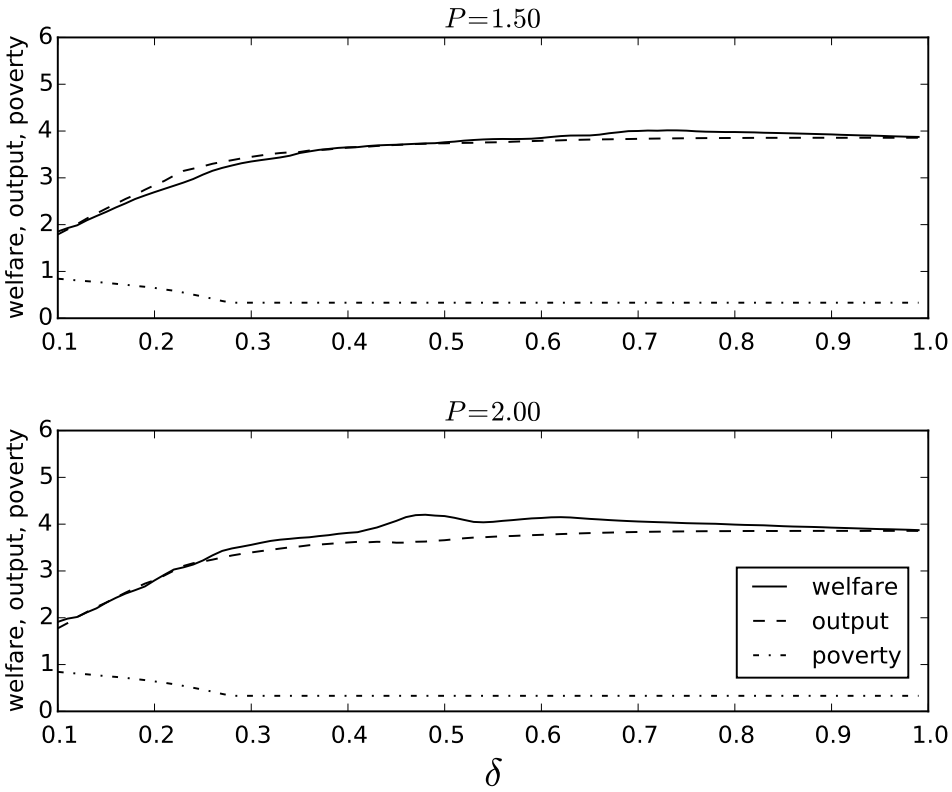

Figure 17: Welfare Effects of the External Price

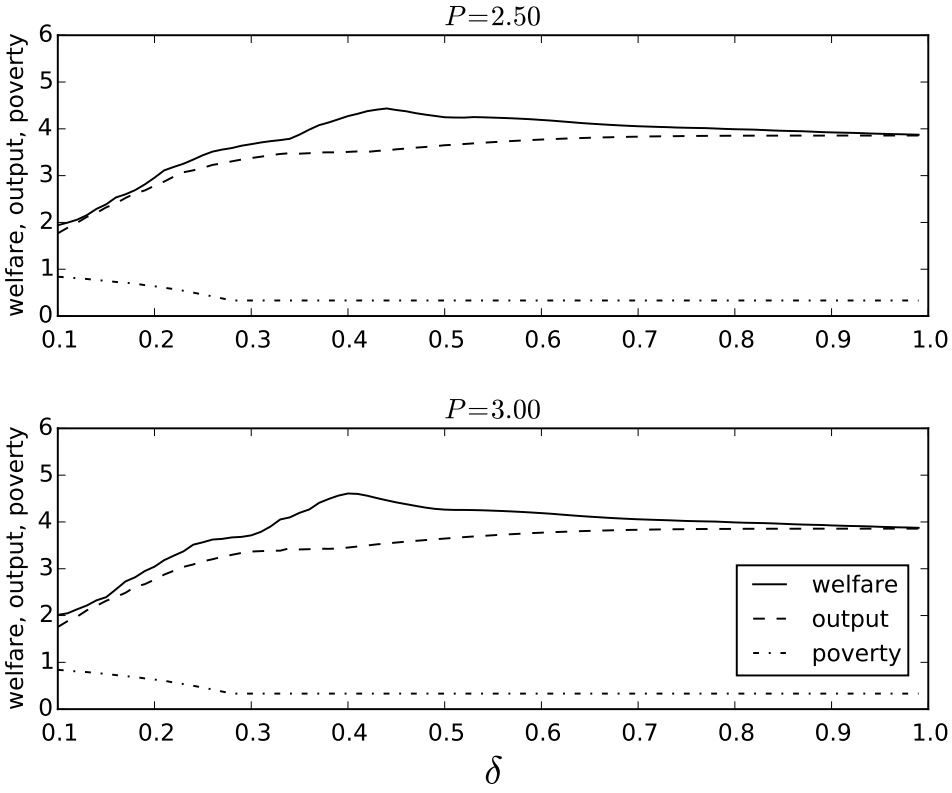

Figure 18: Welfare Effects of the External Price

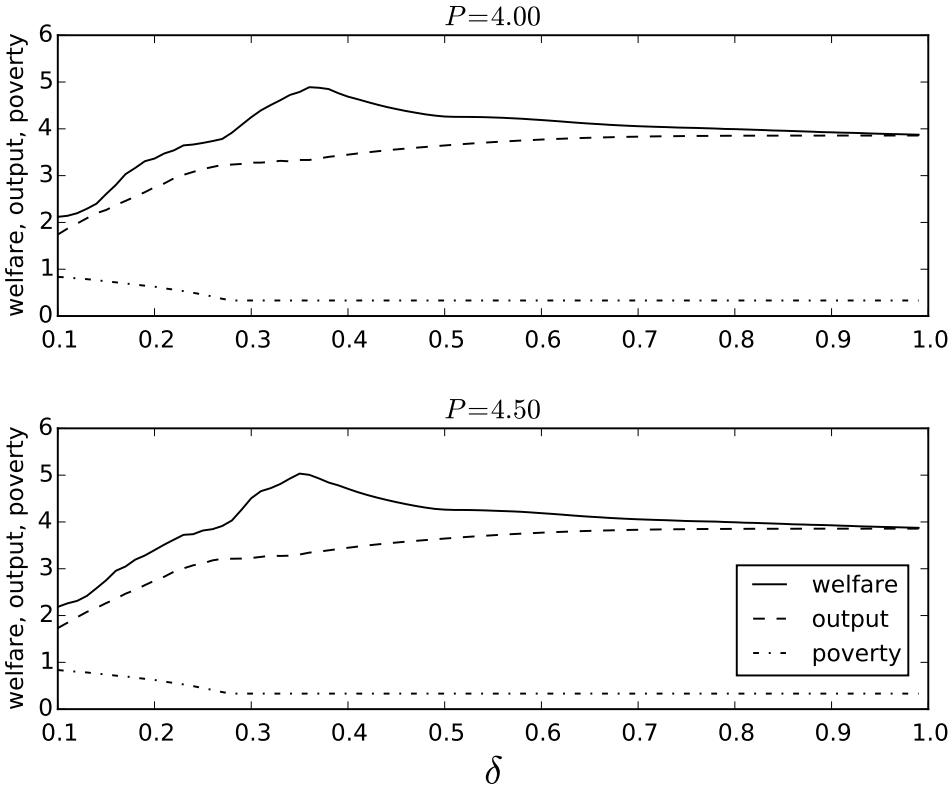

Figure 19: Welfare Effects of the External Price

A.5 Output and Procurement Price Effects of P

The procurer's sales price naturally has large effects on procurer profitability. Figure 20 illustrates somewhat less obvious effects: the response of the procurement price offered by the procurer and the resulting changes in the amount of output destined for the modern sector. We illustrate these outcomes for very large changes in P: halving and doubling the price relative to the baseline value of 3.50. Naturally we always observe $1 < \rho < P$. We see that much of the modern sector's potential to attract production has already been exhausted at the baseline procurer sales price, although a substantial response remains at moderate levels of land inequality. The lower price ($P = 1.75$) is so close to the price in the traditional sector that the procurer has little maneuvering room. As a result, the modern sector attracts less participation, and Q_M is correspondingly low.

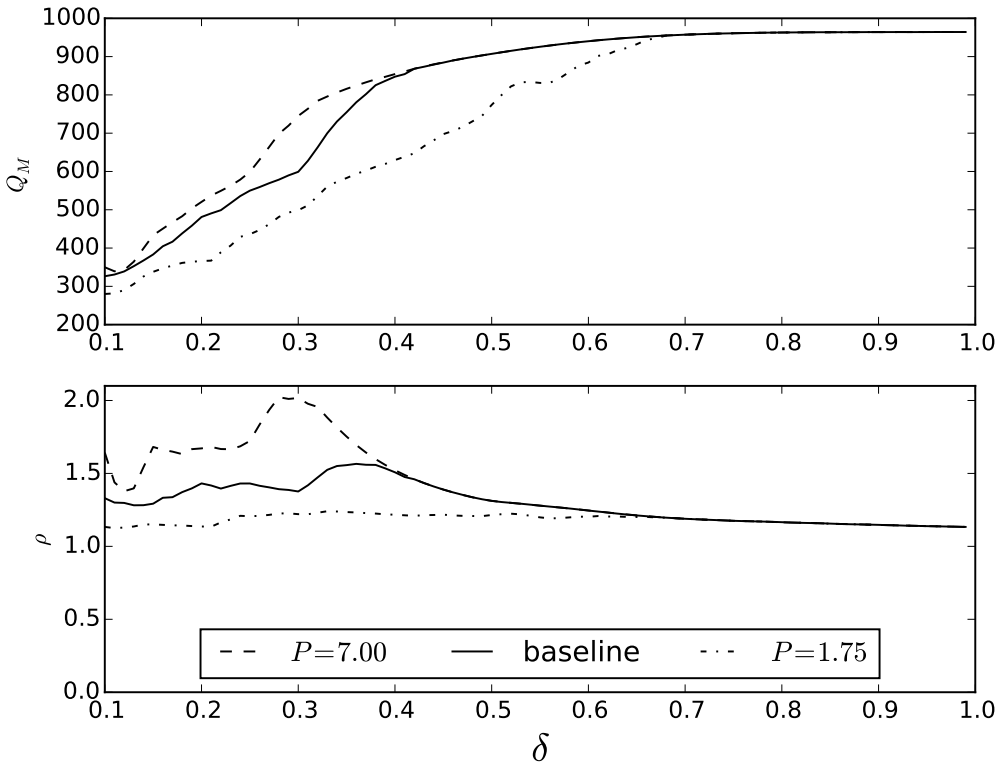

Figure 20: Output and Procurement Price Effects of P

A.6 Sensitivity to K_M

In Figure 9 of the paper, we found that the relationship between modern-sector participation costs (K_M) and producer welfare is complex. Even with substantial variation in K_M, however, no monotonic equity-welfare relationship is recovered.

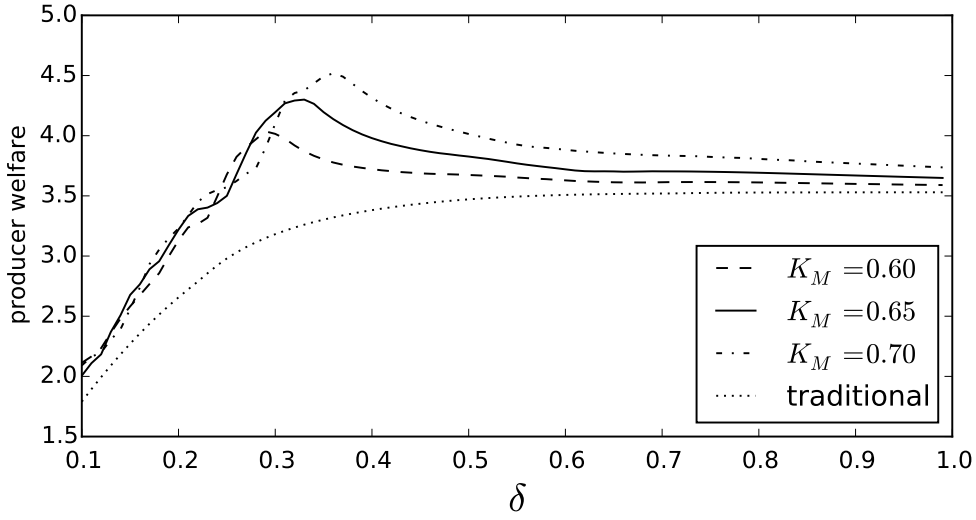

Figure 21: K_M and Producer Welfare

B Baseline Parameterization

```
; File: baseline.ini
; This file specifies the baseline parameters for the simulation in
; "Modern Value Chains and the Organization of Agrarian Production"
; and thus repeats the information in the Table in the paper.
; Here are the EK params for reference (listed as deviations from a
; Figure 2 baseline, as specified in EK's notes to individual figures):
; Baseline:
; Fig 1: Fig 2 but H=0.5 N0=0.0 (note: c not specified)
; Fig 2: A=5,b=0.1,c=0.01,D=0.1,K=0.5,theta=1,phi=0,H=1,N0=0.5,N1=1
; Fig 3: Fig2 plus Yp=1.3
; Fig 4: Fig2
; Fig 5: Fig 2 but b=0.3,c=0.1(typo?),K=0.2,N0=0,N1=1,delta=0.1,Yp=1.0
; Changes in notation
; Us            EK
; --            --
; KT            K
; povertyline   Yp
; p0            N0/(N0+N1)
; s1            b
; s2            c
; Their land is 1 unit with no relative units provided.
; Their labor force is also 1 unit with no relative units provided.
; We match that by scaling H to n_agents (via N1 and N0).
; For our baseline model, we need two additional parameters: KM and PF
; (plus a specification of the number of agents, n_agents).
; A couple of the parameters below are irrelevant for the current
; version of the model but are retained for those who want to explore
; extensions. (See the parameter notes below.)

[A]
type : float
value : 5.0
source : EK (1986, Figure 2, p.493)
description : Productivity parameter on Cobb-Douglas production function
[s1]
type : float
value : 0.30
source : EK (1986, Figure 2, p.493)
description : parameter on first-order aspect of supervision function
note : this is EK's b parameter
[s2]
type : float
value : 0.01
source : EK (1986, Figure 2, p.493)
description : parameter on second-order aspect of supervision function
note : this is EK's c parameter
[D]
type : float
value : 0.10
source : EK (1986, Figure 2, p.493)
description : parameter on sub-utility (of leisure) function
```

```
[cdistscale]
type : float
value : 0.0
description : scales cost per unit of distance traveled
  for agroindustrial firm relative to the gridsize
note : no costs in baseline
[KF]
type : float
value : 0.0
description : fixed costs (per grower) for the agroindustrial procurer
[KM]
type : float
value : 0.75
description : grower fixed cost in modern value chain
note : 50pct higher than KT
[KT]
type : float
value : 0.5
source : EK (1986, Figure 2, p.493)
description : producer fixed cost in traditional value chain
note : this is EK's K
[PF]
type : float
value : 3.5
description : sale price of procurer
[phi]
type : float
value : 0.0
source : EK (1986, Figure 2, p.493)
description : Intercept of working capital function
[povertyline]
type : float
value : 1.3
description : Poverty line
source :  EK (1986, Figure 3, p.494)
note : this is EK's Yp parameter
[p0]
type : float
value : 0.33
source : EK (1986, Figure 2, p.493)
description : proportion of landless agents
note :  equals EK's N0/(N0+N1)
[theta]
type : float
value : 1.0
source : EK (1986, Figure 2, p.493)
description : parameter on land owned in working capital function
note: a larger theta means a looser capital constraint
[n_agents]
type : int
value : 250
description : Number of agents
note : EK's theoretical model has a continuum of workers on an interval;
  an agent-based implementation needs a finite number of agents.
```

```
  (See the sensitivity analysis in the paper.)
[n_locations]
type : int
value : 10
description : the number of locations randomly chosen
  for the procurer to consider when deciding where to locate
note : this parameter ignored if cdistscale is 0,
  as it is in the baseline simulation
[n_replicates]
value : 1
type : int
description : the number of replicates for this scenario
note : the baseline simulation is deterministic
  so replicates aren't needed; reset this parameter
  for extensions with important randomness.
[rndseed]
value : 314
type : int
description : initial seed for random number generation
note : each replicate is assigned its a unique seed based on this
[grid_side]
type : tuple
value : computed
description : shape of the rectangular grid (the land)
note : supply value to override computed value
[productionshocks]
type : bool
value : False
description : set to True to subject producer productivity to stochastic shocks;
  Not currently used because it has very little effect on the results.
[dpp]
type : int
value : 1
description : the amount (in percent) to increment delta_pct
  each iteration; determines the data (and plot) density
[extrasearch]
type : bool
value : False
description : set to True to use random search to
  improve discovery of global max in `rho_epi`
```

C Source Code

```
"""File: distribution.py
Provides Python 2.7 source code for classes used in:
Modern Value Chains and the Organization of Agrarian Production

Classes
-------
AgtInfo : a namedtuple holding core agent attributes
MktInfo : a namedtuple holding a basic market summary
Producer : a class for the specification of agent attributes and behavior
Procurer : a class for the specification of firm attributes and behavior
World : a class for the creation, coordination, and documentation of agents
GUI : a class for GUI set-up (only if desired; not needed to run model)

Dependencies
------------
numpy : http://numpy.scipy.org/
scipy : http://www.scipy.org/
gridworld : http://code.google.com/p/econpy/source/browse/trunk/abm/
    (in the baseline, this is just for some GUI conveniences;
    in extensions, provides a topology)
choose : included
parameters : included
"""
from __future__ import division
#standard library imports:
import itertools, logging, math
from collections import Counter, deque, namedtuple, defaultdict
#commonly used scientific libraries:
import numpy as np
from scipy.optimize import fminbound, fsolve
from scipy.optimize import minpack #for trapping minpack.error
import matplotlib.pyplot as plt
#other imports:
import gridworld
import choose
from utilities import *

__version__ = "2.1.3"

############################## global constants ##############################
############### (these are largely for reporting convenience) ################

############### macro-level variables (see `log_macrodata` for definitions)
macro_datanames = tuple("""
delta_pct n_agents KF PF
KM cdistscale theta
v w rho
H hsumPL hsumLC hsumSC hsumSM hsumLG
Qtotal Qav QMtotal QMav
Ytotal Yav YMtotal YMav
Utotal Uav UMtotal UMav Ulandless
p_qmodern p_ymodern Pi
n_modern n_poor n_contract
n_class0 n_class1 n_class2 n_class3 n_class4
n_sector0 n_sector1 n_sector2
n_frenege n_prenege n_accept n_honor
p_modern p_poor p_contract p_qmodern
```

```
hd_total twd_total thd_total Ld_total tsd_total
gini_y gini_u
""".split())

################ agent-level variables
micro_datanames = tuple("""
delta_pct pclass sector
hbar h_d h th th_d tw tw_d L_d L tr tr_d q
is_poor U_d U Y_d Y contract
""".split())

################ contract-related bit flags
CONTRACTS = dict(
    offer=(1<<0),
    accept=(1<<1),
    fhonor=(1<<2),
    frenege=(1<<3),
    prenege=(1<<4),
    )



############################## classes ####################################

AgtInfo = namedtuple('AgtInfo', 'hbar bbar A')

MktInfo = namedtuple('MktInfo', 'delta_pct, v, w, rho, growers')

class Producer(gridworld.Agent):
    """Provides a producing agent,
    which may be landless or landed, laborer or capitalist.
    Inheriting from `gridworld.Agent` is not important
    for the baseline model but provides GUI convenience.
    """

    def initialize(self):
        """Return None; initalize the producer attributes.
        :side-effects: set hbar, bar; set A (as final value);
            change state of prng (if `productionshocks` is True).
        :note: `initialize` is called by `gridworld.Agent.__init__`
        """
        #the first two attributes are reset each delta by `set_landholdings`
        self.hbar = 0.0 #quantity of land owned
        self.bbar = 0.0 #land based credit
        params = self.params  #the model parameters (not agent specific)
        A = params['A']
        #extension: allow production shocks
        if params['productionshocks']: #False in baseline
            prng = params['prng']  #see parameters.py
            A *= prng.lognormvariate(0,0.01)
        self.A = A
        #arbitrary display conveniences (GUI only)
        if self.world.has_gui:
            self.display(fillcolor='white', shape='square')

    def calc_info(self, v, w, rho, has_contract):
        """Return dict: agent's demands and supplies,
        pclass, sector, factor demands, and welfare.
        :calls: `choose`  (which is called nowhere else)
```

```
    :side-effects: none (does NOT set prdr info!)

    Parameters
    ----------
    v (float) : rent of land (positive float)
    w (float) : wage of labor (positive float)
    rho (float) : price premium of modern sector (positive float);
        must be in [1,PF]
    growers (list[Producer]) : producers that "have contracts"
    loc (int,int) : proposed location of firm; default is `self.position`
    has_contract (bool) : True if agent offered contract

    Note
    ----
    The keys of the returned dict are:
    sector,pclass,tr_d,tw_d,th_d,L_d,h_d,q_d,Y_d,lmda,U_d
    (See the `choose.py` module documentation for definitions.)
    """
    assert (w>0 and v>0), "bad factor prices: v={}, w={}".format(v,w)
    params = self.params  #the model params (not agent specific)
    KM, KT = params['KM'], params['KT']
    #K=0 for pure laborer; see below
    assert (rho>=1 and rho <= params['PF']), "bad rho: {}".format(rho)
    #get the info (dict) for each possible choice
    agent = self.base_info #read-only, just to make a point (and add safety)
    lab = choose.pure_laborer(agent, v, w, params)
    trad = choose.cultivate(agent, params['KT'], v, w, 1.0, params) #trad -> K=KT, rho=1.0
    if has_contract and rho>1:   # Growers under contract, full range of options
        mod = choose.cultivate(agent, params['KM'], v, w, rho, params) #K=KM, rho=rho
        sectors = (lab, trad, mod)
    else:      # Growers not under contract; 'mod' is not available
        sectors = (lab, trad)
    #choose sector to maximize utility, with implied fixed costs
    info = max(sectors, key=lambda s: s['U_d'])
    info['sector'] = sectors.index(info)
    assert "contract" not in info
    info['contract'] = 0
    if has_contract:
        info['contract'] |= CONTRACTS['offer']
    if info['sector'] == 0:
        assert info['K'] == 0
    elif info['sector'] == 1:
        assert info['K'] == KT
    elif info['sector'] == 2:
        assert info['K'] == KM
        assert has_contract
        info['contract'] |= CONTRACTS['accept']
    else:
        raise ValueError("unknown sector".format(info['sector']))
    info['vwrho'] = (v, w, rho)
    info['hbar'] = agent.hbar  #only needed for microdata logging
    #next we record an arbitrary value of lmda for those who choose `lab`
    if info is lab:  # lmda does not exist for laborers; arbitrary replacement
        info['lmda'] = 0
    return info

@property
def base_info(self):
    """Return AgtInfo, the core characteristics of this agent,
```

```
        given the land allocation.
        :called by: `choose`
        :note: this provides all the agent info needed by `choose`
        """
        return AgtInfo(hbar=self.hbar, bbar=self.bbar, A=self.A)


    def qYU(self, info):
        """Return 3-tuple of float,
        the producer's output, income, and utility.
        :note: info must include ex post values
          (which allow for short-side constraints),
          provided by `updated_producer_infos`
        """
        hbar = self.hbar
        assert hbar == info['hbar'] #id's the producer
        #compute q
        h = info['h']
        L = info['L']
        th = info['th']
        #:note: A may be individualized (but not in baseline)
        q = self.A * math.sqrt(h * (th + L))
        #compute Y
        v, w, rho = info['vwrho']
        yv = v * (hbar - info['h'])
        yw = w * (info['tw'] - info['L'])
        if bool(info['contract'] & CONTRACTS['accept']):
            assert bool(info['contract'] & CONTRACTS['offer'])
            price = rho
        else:
            price = 1.0
        yq = price * q
        Y = yq + yv + yw - info['K']  #see `calc_info`
        D = self.params['D']
        U = Y + D * math.sqrt(info['tr'])
        return (q, Y, U)

    ### read-only properties ###

    @property
    def params(self):
        return self.world.params

    #minor safety features

    @property
    def info(self):
        raise NotImplementedError

    # GUI stuff

    def redraw(self, info):
        """Return None.  Reset agent display attributes.
        Side effects: GUI only.
        """
        if not self.world.has_gui:
            return
        sector = info['sector']
        contract = info['contract']
```

```
        fillcolor = ('#FFFFFF','#EEEEFF','#0000FF')[sector] #lab, trd, mod = 0, 1, 2
        if sector==2 and (CONTRACTS['frenege'] & contract):
            fillcolor = '#FF0000'   #color freneged (color choice? chk)
        size = (self.hbar + 0.01)**(1/6.0)  #display size reflects land owndership
        self.display(fillcolor=fillcolor, shape='square', shapesize=(size, size))



class Procurer(gridworld.Agent):
    """Provides a modern-sector firm that can contract with growers.
    Inheriting from `gridworld.Agent` is not important
    for the baseline model but provides GUI convenience.
    """

    def initialize(self):
        """Return None. Set miscellaneous initial values.
        :note: this method is called by gridworld.Agent.__init__
        """
        self.reset()
        #for GUI user convenience, make firm gray until it chooses a location
        if self.world.has_gui:
            self.display(fillcolor='gray', shape='circle', shapesize=(1,1))

    def reset(self):
        """Return None.
        :side-effects: Reset growers attributes to initial state.
        """
        self._growers = None
        self._honored = None
        self._freneged = None
        self.rho = 1
        self.Pi = 0

    def choose_location(self, v, w):
        """Return None.  Set position to profit maximizing location.
        Side effects: only if `cdistscale` > 0, in which chase
        prng state changes (via`random_locations`) if `cdistscale` > 0
        and via `rho_epi` *if* `extrasearch` is True.
        :note: irrelevant to baseline model

        Parameters
        ----------
        v : float
            rent of land
        w : float
            wage of labor
        """
        world = self.world
        params = self.params
        if (params['cdistscale'] == 0):
            msg = """cdistscale==0;
            distance doesn't matter so location doesn't matter;
            we will leave firm at its initial location."""
            world.logger.info(msg)
            return self.position
        #    ^^^^^^
        print("It can take a while for the procurer to choose a location ...")
        #random possible firm locations
```

```
    n_locations = params['n_locations']
    #get a random set of unoccupied firm locations
    locations = world.random_locations(n_locations, exclude=True, prng=params['prng'])
    loc2rho = dict()        #map of locations to best rho (given v,w)
    loc2epi = dict()        #map of locations to max profits (given v,w)
    for loc in locations:   #find best rho for each location
        reg = self.rho_epi(v, w, loc=loc)
        loc2rho[loc] = reg['rho']
        loc2epi[loc] = reg['epi']
    print("Location chosen.")
    return max(loc2epi, key=loc2epi.get) #choose best position

def profitvwgi(self, rho):
    """Return (float,float,float,list[Producer],dict[Producer,dict]):
    profit, v, w, growers, infodicts.  Finds the best set
    of growers for this `rho`, and also the
    associated factor prices.
    :called by: `findeq_modern` (repeatedly)
    :todo: switch growers to tuple?
    """
    v,w = self.world.clear_factor_markets_traditional() #just to initialize
    infodict = self.world.producer_infos(v=v,w=w,rho=rho,growers=())
    best = (0,v,w,(),infodict)
    if (rho <= 1 or rho >= self.params['PF']):
        return best
    #   ^^^^^^
    growers = self.world.producers  #start with all
    grower_list = [] #list of lists
    improved = False
    ct = 0
    while (ct<100): #two or three tries is typical
        if (growers in grower_list):
            print "grower set determined in {} tries".format(ct)
            break
        grower_list.append(growers)
        v,w = self.world.clear_factor_markets(rho, growers)
        #get profits when all `growers` contracts fhonored!
        # along with the preferred growers at v,w, rho (out of all producers)
        #note: do NOT use `growers_and_profit` instead!
        gp = self.profit02(v, w, rho, growers)
        profit = gp['profit'] #profit when *all* accepted contracts fhonored
        infodict = gp['infodict']
        if profit > best[0]:
            best = (profit,v,w,growers,infodict)
            if ct > 0:
                improved = True  #improvement over offer to all
        #now we update to the growers that were preferred at v,w,rho
        #(but this may then lead to factor price changes, next iter)
        growers = gp['growers']
        ct += 1
    if (not improved):
        #but not every producer will end up modern, of course
        #so let's figure out who the growers really are
        growers =  best[-2]
        if (growers == self.world.producers): #offering to everyone is optimal
            profit_best, v_best, w_best, g_best, i_best = best
            growers = tuple(p for (p,i) in i_best.items() if i['sector']==2)
            assert np.allclose(profit_best
                               ,self.profit02(v_best,w_best,rho,growers)['profit']
```

```
                                )
                best = (profit_best, v_best, w_best, growers, i_best)
            else:
                assert growers==() #we never saw positive profits at this rho
        #chk remove the following slightly expensive test? (profitvwgi called in loop...)
        for grower in growers:
            assert infodict[grower]['sector'] == 2
        #print "profitvwgi:", best[:3], rho, len(best[3])
        msg = """At rho={}, we find the best profit={} after {} tries."""
        print msg.format(rho, best[0], ct)
        return best

    def profit02(self, v, w, rho, growers, loc=None):
        """Return dict, mapping (when all `growers` can successfully opt in)
        'profit'->the procurer's profit (at `loc`) &
        'growers'->the procurer's profitable growers out of *all* producers
        :note: profit when all `growers` have their contracts honored,
            so the returned level of profits can be negative.
        :note: Profits are "expected" only in the sense of expecting
          `v` and `w` unchanged.
        :note: it is sensible to call this function
          ONLY when v, w, are market clearing values for rho, growers
        :note: procurer honors contracts (procurer will buys from all `growers`);
          contrast with `growers_and_profit`
        :called-by: `profitvwgi`
        :side-effects: none

        Parameters
        ----------
        v (float) : rent of land (positive float)
        w (float) : wage of labor (positive float)
        rho (float) : price premium of modern sector (positive float)
        growers (list[Producer]) : producers that "have contracts"
        loc (int,int) : proposed location of firm; default is `self.position`
        """
        if not (v>0 and w>0): #precondition
            raise ValueError("factor prices shd be positive")
        params = self.params
        PF, KF = params['PF'], params['KF']
        if (rho <=1): #no growers will accept this
            return dict(profit=0, growers=(), infodict=None)
        #    ^^^^^^
        if loc is None: #:note: location does not matter in the paper simulations
            loc = self.position
        world = self._world
        cdist, producers = world.cdist, world.producers
        if cdist > 0:
            distances = self._world.producer_distances(loc)
        #get producer info when ALL producers have modern option:
        infodict = world.producer_infos(v=v, w=w, rho=rho, growers=producers)
        new_growers = list() #the desirable (profitable) producers
        total_profit = 0
        for prdr in producers: #retain order by using producer list
            info = infodict[prdr]
            contract = info['contract']
            assert (contract & CONTRACTS['offer']) #every prdr has an offer
            #determine the actual profit from the existing contracts
            if info['sector'] == 2: #if prdr wd accept an offer (ow no profit)
                assert (contract & CONTRACTS['accept'])
```

```
            revenue = (PF-rho) * info['q_d']
            cost = KF
            if (cdist > 0): #no distance costs in baseline
                cost += cdist * distances[prdr]
            profit = revenue - cost
            #determine if the procurer would like to contract with prdr
            if (profit > 0):
                new_growers.append(prdr) #a desired grower at v,w,rho
            #this grower would accept a contract offer; check profitability
            if prdr in growers: #a grower (producer with a contract)
                total_profit += profit  #all contracts honored
    return dict(profit=total_profit, growers=new_growers, infodict=infodict)

def rho_epi(self, v, w, loc=None):
    """Return dict,
    'rho'->the best rho,
    'epi' -> expected profit,
    'growers'->growers at (v,w,rho),
    Given the specified location (default: current position),
    where 'best' means greatest expected profit,
    Note: when we consider firm location & positive distance costs,
    this is called repeatedly with *ex ante* `v` and `w`
    (i.e., prior to modern sector) to determine optimal location.
    Side effect: prng state, if extrasearch used;
      see baseline.ini for details.

    Parameters
    ----------
    v : float
        rent of land, should be positive
    w : float
        wage of labor, should be positive
    loc : 2-tuple of int
        procurer location for computation; optional
    """
    if (v < 0 or w < 0): raise ValueError("bad factor prices")
    if loc is None:
        loc = self.position
    params = self.params
    PF = params['PF']       #world price
    profite = self.profite #declare local var -> avoid repeated attribute access
    f2min = lambda p: -profite(v=v, w=w, rho=p, loc=loc)
    #offering rho<1 contracts pointless; rho>PF contracts counterproductive
    results = fminbound(f2min, 1.0, PF, xtol=1e-08,full_output=True,disp=0)
    rho = results[0]  #best premium price for procurer
    EPi = -results[1]  #expected profit
    if __debug__:
        msg = "Best rho={} given v={} and w={} (using fminbound)."
        print msg.format(rho,v,w)

    if rho > PF:
        msg = "Bad fminbound result: rho={}".format(rho)
        raise ValueError(msg)

    #for more deterministic results, extrasearch turned off in the baseline
    extrasearch = self.params['extrasearch']
    if extrasearch: #search nearby for better values
        msg = "rho={} -> profits={}; trying to improve on this..."
        world.logger.info(msg.format(rho,EPi))
```

```
        rvals = [rho]
        pvals = [EPi]
        prng = params['prng']
        for _ in range(25):
            #repeatedly try a random value nearby
            rtemp = 1 + prng.lognormvariate(math.log(rho-1), 0.5)
            ptemp = profite(v, w, rtemp, loc)
            if ptemp > EPi:
                EPi = ptemp
                rho = rtemp
                msg = "Improved by random search\n\t{}->{}".format(rtemp,ptemp)
                world.logger.info(msg)
            #try to do even better with a fitted value
            if ptemp > 0: #collect values for a polynomial fit effort
                rvals.append(rtemp)
                pvals.append(ptemp)
            if len(rvals) > 2:
                a,b,c=np.polyfit(rvals,pvals,2)
                if (a < 0):
                    rtemp = -b/(2*a)
                    ptemp = profite(v, w, rtemp, loc)
                    if ptemp > EPi:
                        EPi = ptemp
                        rho = rtemp
                        msg = "Improved by structured search\n\t{}->{}"
                        msg = msg.format(rtemp,ptemp)
                        rvals.append(rtemp)
                        pvals.append(ptemp)
                        world.logger.info(msg)
    """
    We may need a final function call!
    Why? The chosen rho returned need not be identical to the last rho passed by fminbound,
    so choose again (surprisingly, even w/o extrasearch, this can matter)
    """
    assert EPi == profite(v, w, rho, loc)
    gp = self.growers_and_profit(v, w, rho, loc=loc)
    assert np.allclose(EPi, gp['profit'])
    return dict(rho=rho, epi=EPi, growers=gp['growers'])

def profite(self, v, w, rho, loc=None):
    """Return float, the procurer's maximum expected profit,
    given its location. (Expected only in the sense of expecting
    `v` and `w` unchanged after the contracts are offered,
    which will be true in the general equilibrium.
    :side-effects: none

    Parameters: see `growers_and_profit`

    :note: During setup, this may be called repeatedly
      by `rho_epi` with *provisional* firm locations.
    """
    gp = self.growers_and_profit(v, w, rho, loc=loc)
    return gp['profit']

def growers_and_profit(self, v, w, rho, loc=None):
    """Return dict, mapping (when *all* producers can opt in)
    'profit'->the procurer's maximum expected profit (at `loc`) &
    'growers'->the procurer's chosen (i.e., profitable) growers
    to generate this profit level.
```

```
    Profits are "expected" only in the sense of expecting
    `v` and `w` unchanged after the contracts are offered,
    which will be true in the general equilibrium.
    :note: a set of contracts is NOT imposed;
      contrast with `profit02`
    :side-effects: none

    Parameters
    ----------

    v (float) : rent of land (positive float)
    w (float) : wage of labor (positive float)
    rho (float) : price premium of modern sector (positive float)
    growers (list[Producer]) : producers that "have contracts"
    loc (int,int) : proposed location of firm; default is `self.position`

    Notes
    -----
    Called by `rho_epi` which may be called *repeatedly*
    with *provisional* firm locations.
    """
    if not (v>0 and w>0): #precondition
        raise ValueError("factor prices shd be positive")
    params = self.params
    PF, KF = params['PF'], params['KF']
    if (rho <=1 or rho >= PF):
        return dict(growers=(), profit=0)
    #    ^^^^^^
    if loc is None:
        loc = self.position
    cdist, producers = self._world.cdist, self._world.producers
    if cdist > 0:
        distances = self._world.producer_distances(loc)
    #get producer info when *all* have modern option
    infodict = self._world.producer_infos(v=v, w=w, rho=rho, growers=producers)
    growers = list()  #the desirable (profitable) growers
    total_profit = 0
    for prdr in producers: #determinate order of iteration
        info = infodict[prdr]
        contract = info['contract']
        assert (contract & CONTRACTS['offer'])  #pretended every prdr has an offer
        if info['sector'] == 2: #if prdr wd accept an offer
            assert (contract & CONTRACTS['accept'])
            #this producer wants a contract, check ex ante (KF) profitability
            #to see if it will actually get one
            revenue = (PF-rho) * info['q_d']
            cost = KF
            if (cdist > 0): #no distance costs in baseline
                cost += cdist * distances[prdr]
            profit = revenue - cost
            if (profit > 0):
                total_profit += profit
                growers.append(prdr)
    return dict(profit=total_profit, growers=growers)

def choose_growers(self, v, w, rho):
    """Return list of Producer,
    the producers who appear profitable ex ante,
    given `v`, `w`, `rho`, and the location of the firm.
```

```
    :note: One might expect contract-related fixed costs
        to be incurred here, but that does not make sense
        in the EK model, where fixed costs are covered
        by factor sales at realized factor prices.
    :side-effects: none
    """
    gp = self.growers_and_profit(v, w, rho)
    return gp['growers']


def choose_contracts(self, infodict):
    """Return an infodict, marking the growers whose contracts are honored.
    Determines firm's quantity contracted, profit, and rejected growers.
    :note: Call this function *after* the world clears factor markets
           for firm's chosen rho.
    :note: this function is roughly a no-op in the baseline,
           where all contracts are honored
    :side-effects: none
    """
    world = self.world
    infodict = world.updated_producer_infos(infodict) #new copy
    producers = world.producers  #for determinate iteration order
    params = self.params
    KT = params['KT']
    KM = params['KM']
    PF = params['PF'] #received price per unit of output
    world = self.world
    cdist = world.cdist   #cost per unit distance (zero in baseline case)
    #check the profitability of each contract
    if (cdist > 0):
        distances = world.producer_distances(self.position)
    #Get the *ex post* info dicts, given the specification of growers
    # (i.e., contract offers); includes contract acceptances
    producers = infodict.keys()
    #are these lists useful enough to keep? ... chk
    growers, rejecters, honored, freneged, preneged = [], [], [], [], []
    total_profit = 0
    vwrho = None
    for prdr in producers:
        info = infodict[prdr]
        if vwrho is None:
            vwrho = info['vwrho']
            v, w, rho = vwrho
        else:
            assert vwrho == info['vwrho']
        contract = info['contract']
        sector = info['sector']
        if bool(contract & CONTRACTS['offer']):
            growers.append(prdr)
        if not bool(contract & CONTRACTS['accept']):
            assert info['K'] == (0,KT)[sector]
            if bool(contract):
                assert bool(contract & CONTRACTS['offer'])
                rejecters.append(prdr)
        else: #contract was accepted
            assert info['K'] == KM
            assert bool(contract & CONTRACTS['offer'])
            if sector != 2: #then the grower reneged (cannot happen in baseline)
                preneged.append(prdr)
```

```
                info['contract'] |= CONTRACTS['prenege']
            else: #grower sector is 2, so KM and KF **already** sunk
                #producer wants to sell, check profitability (after cdist and KF)
                revenue = (PF-rho)* info['q']
                cost = 0  #:note: KF already sunk
                if (cdist > 0):
                    cost += cdist * distances[prdr]
                profit = revenue - cost
                if profit > 0:
                    honored.append(prdr)
                    info['contract'] |= CONTRACTS['fhonor']
                    total_profit += profit
                else:
                    freneged.append(prdr)
                    #the next is redundant; just for readers convenience
                    info['contract'] |= CONTRACTS['frenege']
                    info['sector'] = 1  #sector demotion!
    if len(freneged)>0:
        dpct = world.delta_pct
        msg = "delta {0}: {1} contracts freneged\n"
        msg = msg.format(dpct,len(freneged))
        world.logger.info(msg)
        print(msg)
    if len(preneged)>0:
        dpct = world.delta_pct
        msg = "delta {0}: {1} contracts preneged\n"
        msg = msg.format(dpct,len(preneged))
        world.logger.info(msg)
        print(msg)
    assert len(growers) == (len(rejecters) + len(honored)
                            + len(freneged) + len(preneged)), \
        "{} growers but {} rejecters + {} honored + {} freneged + {} preneged".format(
        len(growers),len(rejecters),len(honored),len(freneged), len(preneged))
    self.honored = honored  #property ensures it's only set once
    self.freneged = freneged  #property ensures it's only set once
    return infodict


@property
def freneged(self):
    """list[Producer] : the producers reneged on by procurer.
    (Not in baseline model.)
    """
    return self._freneged

@freneged.setter
def freneged(self, seq):
    if self._freneged is None:
        self._freneged = seq
    else:
        raise ValueError('freneged already set; set once only')

@property
def honored(self):
    """list[Producer] : the producers not reneged on by procurer.
    (Not in baseline model.)
    """
    return self._honored
```

```
    @honored.setter
    def honored(self, seq):
        if self._honored is None:
            self._honored = seq
        else:
            raise ValueError('honored already set; set once only')

    def profitxp(self, rho, infodict):
        """Return float, the *ex post* total profit.
        State dependence: rho, position, honored, growers.
        :todo: discard as redundant?
        :note: sunk costs reduce profitxp
        :called by: log_macrodata
        :side effects: none
        """
        params = self.params
        PF = params['PF'] #received price per unit of output
        KF = params['KF'] #fixed cost per **honored** grower
        cdist = self._world.cdist   #cost per unit distance
        #check the profitability of each contract
        if(cdist > 0):
            distances = self._world.producer_distances(self.position)
        total_profit = 0
        honored = list()
        freneged = list()
        for (prdr, info) in infodict.items():
            profit = 0
            contract = info['contract']
            if bool(contract & CONTRACTS['fhonor']):
                honored.append(prdr)
                assert not bool(contract & CONTRACTS['frenege'])
                assert prdr in self.honored
                profit += (PF - rho) * info['q']
                if cdist > 0:
                    profit -= cdist*distances[prdr]
                profit -= KF
            elif bool(contract & CONTRACTS['frenege']):
                freneged.append(prdr)
                profit -= KF   #:note:
            total_profit += profit
        if bool(honored):
            assert honored == self.honored,\
            "#honored {}, #self.honored {}".format(len(honored),len(self.honored))
        if bool(freneged):
            assert freneged == self.freneged
        return total_profit

    @property
    def params(self):
        return self.world.params

class World(gridworld.GridWorld):
    """Provides a governing agent,
    which creates, coordinates, and documents other agents.
    """
    setup_complete = False
    _params = None
    _procurer = None
    _producers = None
```

```
_producer_distances = None
_prng = None
#delta_pct controls Pareto distn of land; see our paper
delta_pct = 10 #starting value
ticks = 0
#rental rate (v) and wage rate (w) arbitrary starting values:
#(remember: you cannot index a deque with a slice)
vw_init01 = deque(itertools.product([1.1 + i/10. for i in range(10)],
    [0.1 + i/10. for i in range(10)]),100)
vw_init = deque(vw_init01, 100)
rho_init = deque([1.35], 4)  #:note: arbitrary initialization

def setup(self):
    """Return None.
    Do the initial setup of the entire world.
    Call setup_agents and set up log file."""
    self.logger.info('enter World.setup')
    self.setup_complete = False
    print "\nSetup in progress: please wait ..."
    self.set_derivedvals()
    self.setup_agents()
    self.setup_outfiles()
    self.test_setup()
    self.setup_complete = True
    print "setup complete: you can now run the model"
    self.logger.info('exit World.setup')

def set_derivedvals(self):
    """Return None; set up derived values, which are final
    (i.e., these values will remain constant).
    :side-effects: set world attributes N0, N1, H, cdist
    :todo: use properties to enforce finality
    """
    params = self.params
    N = params['n_agents']
    self.N0 = N0 = int(round(N * params['p0'])) #number of landless
    self.N1 = N1 = N - N0                       #number of landed
    self.H = N1//2 + N0 #(discrete population; match EK Fig 1 & 2 params)
    #the following are not relevant to the baseline scenario
    gridsize = self.topology.shape[0]
    cdist = 10. / gridsize  #basic scaling to size of grid
    cdist *= params['cdistscale']  #baseline is 0
    self.cdist = cdist

def setup_agents(self):
    """Return None; create producer agents and procurer firm.
    Note that this setup does NOT include land distribution.
    (See `setup_run` for that.)
    :note: location does not matter in the baseline scenario
    """
    #start out with the firm at the center
    #(if location matters, we'll later relocate the firm to a better spot)
    params = self.params
    center = self.topology.shape[0] // 2
    self._procurer, = self.create_agents(
        AgentType=Procurer,
        number=1,
        locations=[(center,center)]
        )
```

```
    #create producers in random locations (which will remain fixed)
    #(but locations do not even matter in the baseline)
    agents = self.create_agents(
        AgentType=Producer,
        number=params['n_agents']
        )


def schedule(self):
    """Return None. Run one iteration.
    This is the core schedule of the simulation.
    """
    self.logger.debug("enter schedule")
    if not self.setup_complete:
        raise ValueError('must setup before running; please restart')
    # set up the new iteration by resetting all agents
    self.ticks += 1
    delta_pct = self.delta_pct #the land-distribution control parameter
    #setup for this delta: reset agents and distribute land
    self.setup_run(delta_pct/100.) #side effects: via `set_landholdings`
    msg = """
    ############## IMPLEMENT EK MODEL: NO MODERN SECTOR ##############
    EXPERIMENT: {}
    CLEAR MARKETS WITH NO MODERN SECTOR: delta_pct={}
    """.format(self.params.get('name',"unknown"), delta_pct)
    print(msg)
    self.logger.info(msg)
    #start by determining traditional factor market clearing
    vxa, wxa = self.clear_factor_markets_traditional()
    self.add2vwinit(vxa, wxa, 1) #side effect: append to vw_init01
    #log the resulting data (pre-modern, so: rho=1 and no growers)
    infodict = self.producer_infos(v=vxa, w=wxa, rho=1, growers=())
    infodict = self.updated_producer_infos(infodict)
    #naturally, there has not been any contracting (since no modern sector):
    assert not any(info['contract'] for info in infodict.values())
    ## RECORD DATA (with no modern sector present)
    mktinfo = MktInfo(delta_pct=delta_pct, v=vxa, w=wxa, rho=1, growers=())
    self.log_macrodata(self.oldmacro, mktinfo, infodict)
    if (0 == delta_pct % 10): #log micro data every 10th iteration
        self.log_microdata(self.oldmicro, infodict)
    msg = """
    ############## INTRODUCE MODERN SECTOR ##############
    EXPERIMENT: {}
    CLEAR MARKETS WITH MODERN SECTOR: delta_pct={}
    """.format(self.params.get('name',"unknown"), delta_pct)
    print(msg)
    self.logger.info(msg)
    # procurer chooses location
    firm = self.procurer
    if self.ticks==1: #on the first iteration, set firm.position
        firm.position = firm.choose_location(v=vxa, w=wxa) #ex ante!
        self._producer_distances = self.producer_distances(firm.position)
        if self.has_gui:
            firm.display(fillcolor='black', shape='circle', shapesize=(1,1))
    ############   CLEAR MARKETS WITH MODERN SECTOR    ############
    ############ (iterate to find rho,growers,v,w equilibrium) ############
    eq = self.findeq_modern()
    v, w, rho, growers = eq['v'], eq['w'], eq['rho'], eq['growers']
```

```
        firm.rho = rho  #we could get rid of this; it's mostly for GUI ease chk
        """What follows is not relevant to the baseline model.
        In a model where the procurer can behave opportunistically
        (which requires KF > 0, among other things, which it is NOT in baseline),
        we would need to determine which contracts are honored.
        """
        infodict = self.producer_infos(v=v, w=w, rho=rho, growers=growers)
        infodict = firm.choose_contracts(infodict) # calls 'updated_producer_infos'
        self.redraw_producers(infodict)  #just for the gui
        ## RECORD DATA (with modern sector present)
        mktinfo = MktInfo(delta_pct=delta_pct, v=v, w=w, rho=rho, growers=growers)
        self.log_macrodata(self.newmacro, mktinfo, infodict=infodict) #  record macro data
        if (0 == delta_pct % 10): #  record micro data at select delta vals
            self.log_microdata(self.newmicro, infodict)
        """We close out this iteration with small side effects,
        which are just preparation for next iteration."""
        #increment the equality parameter for the next iteration
        self.delta_pct += self.params['dpp']
        #finally, add v,w, rho to initial values for next iteration
        #(use as subsequent initial values, to speed convergence)
        self.rho_init.append(rho)
        self.add2vwinit(v, w, rho)
        if self.delta_pct > 99:
            self.stop()
        self.logger.debug("exit schedule\n")

    def reset(self):
        """Return None; reset the procurer attributes.
        """
        self.procurer.reset()

    def add2vwinit(self, v, w, rho):
        #This is *very* model specific: we know v and w shd be limited,
        #so we don't retry "bad" outcomes
        #:note: new starting values may affect outcomes, so use with caution
        if v < 3 and w < 2:  #don't retry "bad" outcomes
            if(rho==1):
                self.vw_init01.append((v,w))
            else:
                self.vw_init.append((v,w))

    def bracket_maxprofit(self):
        """Return 6-tuple of float (the bracket).
        """
        PF = self.params['PF']
        f2min = lambda p: -self.procurer.profitvwgi(rho=p)[0]
        rho_best = np.mean(self.rho_init)  #somewhat arbitrary starting value
        profit_best = self.procurer.profitvwgi(rho=rho_best)[0]
        #protect against the best being nevertheless bad
        if profit_best <= 0: #in this case we will try a line search
            step = (PF - 1) / 10
            rhos = [1 + step * (i+1) for i in range(9)]
            pvwgis = [self.procurer.profitvwgi(rho) for rho in rhos]
            profits = list(pvwgi[0]  for pvwgi in pvwgis)
            assert all((profit >= 0) for profit in profits)
            idx = profits.index(max(profits))
            profit_best = profits[idx]
            rho_best = rhos[idx]
        if profit_best <= 0: #it does not pay the procurer to operate
```

```
        bracket = None
    else:
        bracket = shrink_bracket(f2min, 1, rho_best, PF, 0, -profit_best, 0)
    return bracket

def findeq_modern(self):
    """Return tuple: v, w, rho, growers characterizing equilibrium:
    rho, growers is the firm's best response to v,w, which in turn
    is a factor market equilibrium.
    :note: offering rho<1 contracts pointless; rho>PF contracts counterproductive,
      so we can bracker the best rho within that interval
    :note: currently prints lots of notes as user info
    :side-effects: calls self.reset()
    """
    self.reset()  #:note: just resets procurer
    bracket = self.bracket_maxprofit()
    if bracket is None: #modern sector is not profitable
        v, w = self.clear_factor_markets_traditional()
        return dict(profit=0, v=v, w=w, rho=1, growers=())
    #    ^^^^^^
    else:
        (xa, xm, xb, fa, fm, fb) = bracket
    rho_best = xm
    profit_best, v_best, w_best, growers_best, infodict_best =\
        self.procurer.profitvwgi(rho=rho_best) #initialize
    #Remove the following redundant test? chk
    for grower in growers_best:
        sector = infodict_best[grower]['sector']
        assert  sector== 2, "{} vs {}".format(sector,2)
    assert np.allclose(-fm,profit_best)
    print """NOTE: SEARCH for equilibrium with modern sector.
    Our initial rho value (based on previous rho values) is: {}.
    This rho gives us a profit of {}.
    (We'll try to do better than that.)  BEGIN SEARCH ...
    """.format(rho_best, profit_best)
    f2min = lambda p: -self.procurer.profitvwgi(rho=p)[0]
    print """NOTE: starting bracket is ({},{},{});
    starting fvals are ({},{},{})
    """.format(xa,xm,xb,fa,fm,fb)
    #next do the simplest imaginable bracket search for the best rho
    rho, negprofit = simple_bracket_search(f2min, xa, xm, xb, fa, fm, fb)
    profit, v, w, growers, infodict = self.procurer.profitvwgi(rho=rho)
    #:note: just some error checks
    assert profit == -negprofit, "{} vs {}".format(profit,-negprofit)
    assert profit >= profit_best, "{} vs. {}".format(profit,profit_best)
    #
    if profit > profit_best:
        profit_best, v_best, w_best, rho_best, growers_best, infodict_best = \
            profit, v, w, rho, growers, infodict
    msg = "delta_pct {}: procurer chooses rho={}".format(self.delta_pct,rho_best)
    self.logger.info(msg)
    print(msg)
    return dict(v=v_best
                ,w=w_best
                ,rho=rho_best
                ,growers=growers_best
                ,profit=profit_best
                ,infodict=infodict_best
                )
```

```
def setup_run(self, delta):
    """Return None. Reset the firm and producers.
    Distribute land among producers.
    This prepares for one run (i.e., one land distribution).
    :side-effects: reset firm & producers; distribute land
    """
    #setup firm
    self.procurer.reset()
    #setup producers
    producers = self.producers
    #new land distribution!
    self.set_landholdings(producers, self.delta_pct/100.)
    self.test_runready()

def set_landholdings(self, producers, delta):
    """Return None, set the hbar and bbar of all `producers`.
    Set the display size of the producers, based on land holdings.
    :called-by: setup_run
    :side-effects: set hbar and bbar for each producer
    """
    if not (delta > 0.0) and (delta < 1.0): #preconditions
        raise ValueError("delta={} but must be in (0,1)".format(delta))
    H, N1 = self.H, self.N1
    params = self.params
    phi, theta = params['phi'], params['theta']
    ishares = incremental_shares_pareto(N1, (1-delta) / (1+delta))
    p = 0  #initialize proportion of landed producers
    for i, prdr in enumerate(producers):  #prdr doesn't need to know its p
        if i < N1:
            share = ishares[i]
            hbar = H * share
        else:
            p = 0
            hbar = 0 #prdr is landless
        prdr.hbar = hbar
        #Calculate bbar up front instead of each of the many times needed.
        #in principle, landless could access credit (but, phi=0 in EK)
        prdr.bbar = phi + (theta * hbar)
        #display size will depend on land ownership
        if self.has_gui:
            prdr.display(shapesize=(share + 0.2, p + 0.2))

def clear_factor_markets_traditional(self):
    """Return 2-tuple of float, the equilibrium factor prices
    when there is no modern sector.
    :side-effects: none
    Parameters
    ----------
    add2vwinit : bool
      whether to append to vw_init01 (breaking referential transparency)
    """
    return self.clear_factor_markets(rho=1, growers=())

def clear_factor_markets(self, rho, growers):
    """Return None. Simultaneous factor market clearing.
    :side-effects: none
    Parameters
    ----------
```

```
    rho : float
      relative procurer price
    growers : sequence
      procurers who are offered contracts
    """
    logger = self.logger
    logger.debug("World: Enter clear_factor_markets_simultaneously.")
    #define factor market clearing:
    f2solve = lambda vw: self.xss_land_labor(vw=vw, rho=rho, growers=growers)
    success = False   #we'll set this to True if we get convergence
    if(rho==1):
        vw_init = self.vw_init01
    else:
        assert rho > 1
        vw_init = self.vw_init
    for ct, vw in enumerate(reversed(vw_init)): #try stored starting values
        try: #comment: tighter xtol does not help
            output = fsolve(f2solve, vw, full_output=1, xtol=1e-06)
            (v, w) = map(abs, output[0]) #needed because xss_land_labor uses abs
        except minpack.error: #factor prices should stay positive
            continue  #try another set of initial values
        if output[2]==1:
            (z1,z2) = output[1]['fvec']
            if (abs(z1 <1) and abs(z2 < 1)):
                success = True
                break

    #the rest is just reporting with one exception: may append to vw_init
    msg = "Given rho={:6.4f}, ".format(rho)
    if success:
        msg += "factor markets clear at v={:6.4f}, w={:6.4f}.".format(v, w)
        msg += "\n(factor mkt convergence in {} tries)".format(ct+1)
        logger.debug(msg)
        print(msg)
    else:
        msg += '\nfactor market fails to converge:\n' + output[-1]
        logger.warn(msg)
        print(msg)
    self.logger.debug("World: Exit clear_factor_markets_simultaneously.")
    assert (v > 0 and w > 0)
    return v, w

def producer_distances(self, loc):
    """Return dict, mapping agents to floats.
    The distances of the agents from `loc`.
    Memoized: past computations stored by this.
    :side-effects: none.
    """
    #first check if we have already stored the distances:
    distances = self._producer_distances
    if distances is None:
        producers = self.producers
        distances = dict()
        fx, fy = loc
        for prdr in producers:
            ax, ay = prdr.position
            distances[prdr] = math.hypot(fx-ax, fy-ay)
    else:
        assert loc == self.procurer.position
```

```
    return distances


def producer_infos(self, v, w, rho, growers):
    """Return dict, mapping producers to their info.
    :note: expensive, because `calc_info` calls `choose` for each producer
    :side-effects: none
    """
    result = dict()
    for prdr in self.producers:
        is_grower = prdr in growers
        info = prdr.calc_info(v=v, w=w, rho=rho, has_contract=is_grower)
        contract = info['contract']
        assert is_grower == bool(contract & CONTRACTS['offer'])
        assert info['vwrho'] == (v, w, rho)
        result[prdr] = info
    return result


def updated_producer_infos(self, infodict):
    """Return dict, mapping producers to properties
    WITH ex post short-side constraints imposed (see below)
    :called-by: `schedule` and `choose_contracts`
    :side-effects: none
    """
    infodict = infodict.copy()  #avoid side effects
    infoseq = infodict.values()
    producers = self.producers
    #We seem usually to succeed in clearing factor markets,
    # but just in case: if not, take the short side.
    #LAND market
    land_supply, land_demand = self.sd_land(infoseq)
    assert (np.allclose(self.H,land_supply) and land_demand >= 0)
    if (land_supply >= land_demand):
        for info in infoseq:
            assert "h" not in info
            info['h'] = info['h_d']
    else: # xss_land < 0, excess demand for land
        scale = land_supply / land_demand
        assert (0 < scale < 1)
        for info in infoseq:
            assert "h" not in info
            #:todo: might be nice to attend to hbar?
            info['h'] = scale * info['h_d']
    #LABOR market
    labor_supply, labor_demand = self.sd_labor(infoseq)
    #print "sd L:", labor_supply, labor_demand
    assert (labor_supply >= 0 and labor_demand >= 0)
    #:note: we will allow for a small approximation error:
    if abs(labor_supply - labor_demand) < 0.01*self.H:
        for info in infoseq:
            assert "L" not in info
            info['L'] = info['L_d']
            info['tw'] = info['tw_d']
    elif labor_supply < labor_demand: #excess demand for labor
        scale = labor_supply / labor_demand
        assert (0 <= scale < 1)
        for info in infoseq:
            assert "L" not in info
```

```
            info['tw'] = info['tw_d']
            info['L'] = scale * info['L_d']
    else: #excess supply of labor
        scale = labor_demand / labor_supply
        assert (0 <= scale < 1)
        for info in infoseq:
            assert "L" not in info
            assert "tw" not in info
            info['tw'] = scale * info['tw_d']
            print "scaled:", scale, info['tw_d'], info['tw']
            info['L'] = info['L_d']
    params = self.params
    D, s1, s2 = params['D'], params['s1'], params['s2']
    povertyline = params['povertyline']
    for prdr, info  in infodict.items():
        L = info['L']
        info['ts'] = ts = (s1 * L) + (s2 * L * L)
        info['th'] = th = info['th_d']
        info['tr'] = tr = 1 - info['tw'] - th - ts
        assert ("q" not in info) and ("Y" not in info) and ("U" not in info)
        info['q'], info['Y'], info['U'] = q, Y, U = prdr.qYU(info)
        assert np.allclose(info['U'], info['Y'] + D * math.sqrt(info['tr']) )
        info['is_poor'] = int(Y < povertyline)
    self.logger.debug("World: Exit updated_producer_infos.")
    return infodict

def xss_land_labor(self, vw, rho, growers):
    """Return 2-tuple of float: the excess supplies.
    Called repeatedly to clear factor markets (given `rho`, `growers`).

    vw : 2-tuple of float
        land rental price, wage
    rho : float
        modern sector price premium
    :side-effects: none
    """
    v, w = vw
    #constrain v>0, w>0
    v, w = abs(v), abs(w)
    infodict = self.producer_infos(v=v, w=w, rho=rho, growers=growers)
    infoseq = tuple(infodict.values())
    xss_land = self.xss_land(infoseq)
    xss_labor = self.xss_labor(infoseq)
    return xss_land, xss_labor

def xss_land(self, infoseq):
    """Return float, the excess supply of land.
    :side-effects: none
    """
    land_supply, land_demand = self.sd_land(infoseq)
    return self.xss(land_supply, land_demand)

def xss_labor(self, infoseq):
    """Return float, the excess supply of labor.
    :side-effects: none
    """
    labor_supply, labor_demand = self.sd_labor(infoseq)
    return self.xss(labor_supply, labor_demand)
```

```
def sd_land(self, infoseq):
    """Return float, the excess supply of land.

    Parameters
    ----------
    infoseq : tuple of dict
        producers info for all producers
    :side-effects: none
    """
    land_supply = self.H
    land_demand = sum(info['h_d'] for info in infoseq)
    return (land_supply, land_demand)

def sd_labor(self, infoseq):
    """Return float, the excess supply of labor.

    Parameters
    ----------
    infoseq : tuple of dict
        info dict for each producer
    :side-effects: none
    """
    labor_supply = sum(info['tw_d'] for info in infoseq)
    labor_demand = sum(info['L_d'] for info in infoseq)
    return (labor_supply, labor_demand)

@staticmethod
def xss(s, d):
    """Return float, the (possibly scaled) value of s-d.
    Checks for non-negativity.
    """
    assert (d >= 0 and s >= 0)
    if (d==0 and s==0):
        msg = "WARN:\n"
        msg += 'supply: {0}; demand: {1}'.format(s, d)
        print(msg)
        logging.warn(msg)
        result = 0
    elif (d==0 or s==0):
        #result = 2*(s-d)/(s+d)
        result = s - d
    else:
        result = s - d
    return result

def market_info(self, infoseq, v, w):
    """Return str, a market description.
    infoseq : list of dict
      each dict is the info for a producer
    """
    info = "Market info at v={v} and w={w}.".format(v=v,w=w)
    producers = self.producers
    land_supply = sum(prdr.hbar for prdr in producers)
    assert np.allclose(land_supply,self.H)
    land_demand = sum(prdr['h_d'] for prdr in infoseq)
    info += """\nLand Market:
    xs supply:    {hxss}
    (scaled xss:{mhxss})
    land supply:  {hs}
```

```
        land demand:  {h_d}
        n suppliers:  {hsupp}
        n demanders:  {hdmnd}
        """.format(
        hs=land_supply,
        h_d=land_demand,
        hxss=land_supply-land_demand,
        mhxss=self.xss(land_supply,land_demand),
        hsupp=sum(prdr.hbar>0 for prdr in producers),
        hdmnd=sum(prdr['h_d']>0 for prdr in infoseq)
        )
        labor_supply = sum(prdr['tw_d'] for prdr in infoseq)
        labor_demand = sum(prdr['L_d'] for prdr in infoseq)
        info += """
        Labor Market:
        xs supply:    {nxss}
        (scaled xss:{mnxss})
        labor supply: {ns}
        labor demand: {nd}
        n suppliers:  {nsupp}
        n demanders:  {ndmnd}
        """.format(
        ns=labor_supply,
        nd=labor_demand,
        nxss=labor_supply-labor_demand,
        mnxss=self.xss(labor_supply,labor_demand),
        nsupp=sum(prdr['tw_d']>0 for prdr in infoseq),
        ndmnd=sum(prdr['L_d']>0 for prdr in infoseq)
        )
        return info

    def setup_outfiles(self):
        """Return None.  Set up the output files.
        There are four core output files:
        `oldmacro` and `oldmicro` record data from
        the model without a modern sector,
        `newmacro` and `newmicro` record data from
        the model with a modern sector.
        """
        params = self.params
        self.oldmacro = params['oldmacro']
        self.oldmicro = params['oldmicro']
        self.newmacro = params['newmacro']
        self.newmicro = params['newmicro']

        """Add a header to the output files."""
        with open(self.oldmacro, 'w') as fout:
            fout.write( ','.join(macro_datanames) )
        with open(self.oldmicro, 'w') as fout:
            fout.write( ','.join(micro_datanames) )
        with open(self.newmacro, 'w') as fout:
            fout.write( ','.join(macro_datanames) )
        with open(self.newmicro, 'w') as fout:
            fout.write( ','.join(micro_datanames) )

    def log_macrodata(self, fname, mktinfo, infodict):
        """Return None; log macro data from the simulation.
        :todo: streamline accumulated cruft
        :side-effects: none
```

```
    """
    params = self.params
    KM = params['KM']
    KF = params['KF']
    PF = params['PF']
    n_agents = params['n_agents']
    cdistscale = params['cdistscale']
    theta = params['theta']
    povertyline = params['povertyline']
    H = self.H
    #
    delta_pct = mktinfo.delta_pct
    v = mktinfo.v
    w = mktinfo.w
    rho = mktinfo.rho
    growers = mktinfo.growers
    prdrs = list(infodict.keys())
    assert len(prdrs) == n_agents
    assert set(self.producers) == set(prdrs)
    for prdr, info in infodict.items():
        assert (v,w,rho) == info['vwrho'],\
        "v {}, w {}, rho {}, vwrho {}".format(v,w,rho,info['vwrho'])
        Y = info['Y']
        if (Y==0):
            assert (info['K']==0 and info['sector']==0)
    infoseq = infodict.values()
    if (rho==1):
        assert all(info['contract'] == 0 for info in infodict.values())
    #record parameter values
    firm = self.procurer
    delta = delta_pct / 100.
    #class and sector counts
    land_use = defaultdict(float)
    class_counts = defaultdict(int)
    sector_counts = defaultdict(int)
    Ulandless = 0
    n_modern, n_poor = 0, 0
    modern_sector_agents = list()
    n_contract = 0  #number offered contracts
    n_accept = 0  #number accepted contracts
    Qtotal, QMtotal = 0, 0
    Ytotal, YMtotal = 0, 0
    Utotal, UMtotal = 0, 0
    for prdr, info in infodict.items():
        Qtotal += info['q']
        Ytotal += info['Y']
        Utotal += info['U']
        land_use[info['pclass']] += info['h']
        class_counts[info['pclass']] += 1
        sector_counts[info['sector']] += 1
        if prdr.hbar == 0:
            assert info['hbar'] == 0
            Ulandless += info['U']
        if bool(info['contract'] & CONTRACTS['offer']):
            n_contract += 1
        if bool(info['contract'] & CONTRACTS['accept']):
            assert bool(info['contract'] & CONTRACTS['offer'])
            n_accept += 1
        if info['sector'] == 2:
```

```
            QMtotal += info['q']  #modern sector output
            YMtotal += info['Y']  #modern sector incomes
            UMtotal += info['U']  #modern sector incomes
            modern_sector_agents.append(prdr)
            n_modern += 1
        n_poor += info['is_poor']
    assert n_agents == sum(class_counts.values())
    Qav = Qtotal / float(n_agents)
    Yav = Ytotal / float(n_agents)
    Uav = Utotal / float(n_agents)
    assert n_modern == len(modern_sector_agents)
    QMav = 0 if (n_modern==0) else QMtotal / float(n_modern)
    YMav = 0 if (n_modern==0) else YMtotal / float(n_modern)
    UMav = 0 if (n_modern==0) else UMtotal / float(n_modern)
    #class counts
    n_class0 = class_counts[0]
    n_class1 = class_counts[1]
    n_class2 = class_counts[2]
    n_class3 = class_counts[3]
    n_class4 = class_counts[4]
    assert n_class0 + n_class1 + n_class2 + n_class3 + n_class4 == n_agents,\
        "Instead of equal values, we find {} and {}.".format(ct,class_counts.values())
    #sector counts
    n_sector0 = sector_counts[0]
    n_sector1 = sector_counts[1]
    n_sector2 = sector_counts[2]
    assert n_sector0 + n_sector1 + n_sector2 == sum(sector_counts.values()) == n_agents,\
        "Instead of equal values, we find {} and {}.".format(ct,sector_counts.values())
    #land use
    assert np.allclose(H, sum(prdr.hbar for prdr in prdrs))
    assert np.allclose(H, sum(info['hbar'] for info in infoseq))
    hsumPL = sum(info['h'] for info in infoseq if info['pclass']==0)
    hsumLC = sum(info['h'] for info in infoseq if info['pclass']==1)
    hsumSC = sum(info['h'] for info in infoseq if info['pclass']==2)
    hsumSM = sum(info['h'] for info in infoseq if info['pclass']==3)
    hsumLG = sum(info['h'] for info in infoseq if info['pclass']==4)
    if params['phi'] <= 0:
        assert np.allclose(0, hsumPL)
    assert np.allclose((hsumPL,hsumLC,hsumSC,hsumSM,hsumLG),
        (land_use[0],land_use[1],land_use[2],land_use[3],land_use[4]))
    if not np.allclose(H, hsumPL+hsumLC+hsumSC+hsumSM+hsumLG):
        msg = "land use = {0} but land supply = {1}"
        msg = msg.format(hsumLC+hsumSC+hsumSM+hsumLG,H)
        self.logger.warn(msg)
    hd_total = sum(info['h_d'] for info in infoseq)
    twd_total = sum(info['tw_d'] for info in infoseq)
    thd_total = sum(info['th_d'] for info in infoseq)
    Ld_total = sum(info['L_d'] for info in infoseq)
    s1, s2 = params['s1'], params['s2']
    tsd_total01 = sum(s1*info['L_d'] for info in infoseq)
    tsd_total02 = sum(s2*info['L_d']**2 for info in infoseq)
    tsd_total = tsd_total01 + tsd_total02
    assert QMtotal == sum(info['q'] for info in infoseq if info['sector']==2)
    p_qmodern = QMtotal/Qtotal
    Ys = list(info['Y'] for info in infoseq)
    Y = sum(Ys)
    gini_y = gini(Ys)
    p_ymodern = YMtotal / Y
    Us = list(info['U'] for info in infodict.values())
```

```
    gini_u = gini(Us)
    assert all(U > 0 for U in Us)
    p_poor = float(n_poor) / float(n_agents)   #casts for reader convenience
    ms_counts = Counter(infodict[a]['pclass'] for a in modern_sector_agents)
    self.logger.info('{0} modern agents, class: {1}'.format(n_modern,ms_counts))
    n_modern = len(modern_sector_agents)
    p_modern = float(n_modern) / float(n_agents)   #casts for reader convenience
    n_honor = sum(bool(info['contract'] & CONTRACTS['fhonor']) for info in infoseq)
    n_prenege = sum(bool(info['contract'] & CONTRACTS['prenege']) for info in infoseq)
    n_frenege = sum(bool(info['contract'] & CONTRACTS['frenege']) for info in infoseq)
    print "contracts {}, modern {}, frenege {}, prenege {}, accept {}".format(
        n_contract, n_modern, n_frenege, n_prenege, n_accept)
    assert (n_contract==sum(bool(info['contract']) for info in infoseq))
    p_contract = float(n_contract) / float(n_agents)  #cast for reader convenience
    assert rho == firm.rho  #:todo: remove
    Pi = firm.profitxp(rho, infodict)
    if __debug__ and (rho==1.0):
        assert (Pi == 0), "Pi shd be zero (since rho=1) but Pi={}".format(Pi)
    with open(fname, 'a') as fout:
        strdata = [str(eval(x)) for x in macro_datanames]
        fout.write('\n' + ','.join(strdata))

def log_microdata(self, fname, infodict):
    """Return None.
    Log micro data from the simulation."""
    params = self.params
    attnames = micro_datanames
    assert attnames[0]=='delta_pct'
    with open(fname, 'a') as fout:
        for prdr, info in infodict.items():
            strdata = [str(self.delta_pct)]
            strdata += list(str(info[attr]) for attr in attnames[1:])
            fout.write('\n' + ','.join(strdata))

def test_setup(self):
    """Return None. Basic tests of overall setup.
    Also see `test_runready`.
    """
    params = self.params
    #agents
    assert len(self.producers) == params['n_agents'], 'shd match'
    #firm
    firm = self.procurer
    assert firm.params is params
    assert firm.world is self
    #the next test is a bit expensive and has low payoff; remove? chk
    infodict = self.producer_infos(v=1,w=1,rho=1, growers=())
    assert len(infodict) == params['n_agents'], 'shd match'
    assert all(info['sector']<2 for info in infodict.values())

def test_runready(self):
    """Return None.
    """
    params = self.params
    producers = self.producers
    assert len(producers) == params['n_agents'], 'shd match'
    assert np.allclose(self.H,sum(a.hbar for a in producers)),\
        "H={0}; sum={1}".format(self.H, sum(a.hbar for a in producers))
    landless = [a for a in producers if a.hbar==0]
```

```
        assert len(landless)==self.N0, "{} vs {}".format(len(landless),self.N0)
        firm = self.procurer
        assert firm._growers is None
        assert firm._honored is None
        assert firm.rho == 1

    #properties

    @property
    def params(self):
        """Return dict, the model parameters.
        """
        return self._params

    @params.setter
    def params(self, prms):
        """Sets the `params` property.
        Used only once per simulation (during setup).
        """
        if self._params is None:
            self._params = prms
        else:
            raise ValueError('params can only be set once')

    #read-only properties

    @property
    def procurer(self):
        """Return Procurer.
        """
        if self._procurer is None:
            self._procurer, = self.get_agents(Procurer)
        return self._procurer

    @property
    def producers(self):
        """Return tuple of Producer.
        """
        if self._producers is None:
            self._producers = tuple(self.get_agents(Producer))
        return self._producers

    #minor safety features

    @property
    def v(self):
        raise NotImplementedError

    @property
    def w(self):
        raise NotImplementedError

    #GUI stuff:

    def redraw_producers(self, infodict):
        """Return None. Redraws producers.
        GUI relevance only.
        """
        if not self.has_gui:
```

```
            return
        for prdr, info in infodict.items():
            prdr.redraw(info)


class GUI(gridworld.GridWorldGUI):
    """Provides a GUI for the model run.
    For user convenience only.
    """
    def gui(self):
        """Display buttons and plots."""
        self.add_button('Set Up', 'setup')
        self.add_button('Run', 'run')   #run calls schedule (see gridworld.py)
        self.add_button('Stop', 'stop')


# vim: tabstop=4 expandtab shiftwidth=4 softtabstop=4 autoindent
```

```
"""File: utilities.py
Provides various utilities to distribution.py
"""
from __future__ import division
import numpy as np

def gini(x):
    """Return float, the computed Gini coefficient.
    We compute the Gini by approximating the area B
    under the Lorenz curve with a trapezoidal
    Riemann sum. (Right and left sums differ by 1/N).
    """
    xsort = sorted(x) # increasing order
    y = np.cumsum(xsort)
    N = float(len(xsort))
    #approximate the area under Lorenz curve:
    B = sum(y) / (y[-1] * N)
    return 1 + 1./N - 2*B


def incremental_shares_pareto(n, gini):
    """Return list of float, the incremental shares.
    n : int
      number of agents to recieve share
    gini : float
      inequality parameter
    Notes
    -----
    Given the target Gini inequality coefficient g,
    let delta=(1-g)/(1+g). Then the Lorenz curve
    for a Pareto distribution with Gini=g can be written as
    $F(p) = 1 - (1 - p)^\delta$ where $0 < \delta \leq 1$
    and $p$ is the cumulative proportion agents (out of n agents).
    In the discrete case, let $p_i= i/n$ for $i=1,\dots,n$.
    The cumulative share of those with no more than the $i$-th agent is
    $F(p_i) = 1 - (1 - p_i)^\delta$, so the incremental share agent $i$ is
    $F(p_i) - F(p_{i-1})
    =1 - (1 - p_i)^\delta- [1 - (1 - p_{i-1})^\delta]
    = - (1 - p_i)^\delta  + (1 - p_{i-1})^\delta
    = (1 - p_{i-1})^\delta - (1 - p_i)^\delta$
    """
    delta = (1 - gini) / (1 + gini)
    ishares = list()
    p = 0.0
    for i_1 in range(n):
        p_1 = p              # =i_1 / n
        p = (i_1 + 1.0) / n
        ishare = ((1-p_1)**delta - (1-p)**delta)  #incremental share
        ishares.append(ishare)
    return ishares

def shrink_bracket(f, xa, xm, xb, fa, fm, fb):
    """Return tuple, a narrower bracket of the min,
    (xa,xm,xb,fa,fm,fb).
    f : callable
        the function to be minimized
    xa, xm, xb : float
        bracket of the minimizer, with xa < xm < xb
    fa, fm, fb : float
```

```
        values of f at xa. xm, xb
    """
    if not (xa < xm < xb): raise ValueError("not an interval")
    if not (fm < fa and fm < fb):
        msg = "({},{},{})->({},{},{}) is not a bracket"
        msg = msg.format(xa,xm,xb,fa,fm,fb)
        raise ValueError(msg)
    xstart = xm
    fstart = fm
    print f(9*xm/10.)
    print "bracket01: {} -> {} vs {}".format(xm,fm,f(xm))
    #we'll first search to the left
    for wt in range(2,10): #rising weight on boundary
        wt /= 10.0
        xleft = wt*xa + (1-wt)*xstart #move left relative to fixed start value
        fleft = f(xleft)
        if fleft > fm:
            xa, fa = xleft, fleft
            break
        if fleft < fm: #better reference point
            xb, fb = xm, fm
            xm, fm = xleft, fleft
    print "bracket: {} -> {} vs {}".format(xm,fm, f(xm))
    assert (xm <= xstart)
    assert (fm <= fstart)
    #then we'll search to the right
    xstart = xm
    fstart = fm
    for wt in range(2,10): #rising weight on boundary
        wt /= 10.0
        xright = wt*xb + (1-wt)*xstart #keep the movement relative to fixed xstart
        fright = f(xright)
        if fright > fm:
            xb, fb = xright, fright
            break
        if fright < fm: #better reference point
            xa, fa = xm, fm
            xm, fm = xright, fright
    assert (xm >= xstart)
    assert (fm <= fstart)
    return xa, xm, xb, fa, fm, fb

def simple_bracket_search(f, xa, xm, xb, fa, fm, fb):
    """Return float, the approximate minimizer
    found by bracketed search.
    """
    assert xa < xm < xb
    assert fa > fm and fb > fm
    while abs(xb-xa) > 0.005:
        fm_old = fm
        #first see if we can move min to left
        xam = (xa + xm) / 2.
        fam = f(xam)
        if (fam < fm):
            print "bracket: move min to left"
            xb, fb = xm, fm
            xm, fm = xam, fam
            assert fm < fm_old
            assert fm == f(xam)
```

```
            continue
        #next see if we can move min to right
        xbm = (xb + xm) / 2.
        fbm = f(xbm)
        if (fbm < fm):
            print "bracket: move min to right"
            xa, fa = xm, fm
            xm, fm = xbm, fbm
            assert fm < fm_old
            assert fm == f(xbm)
            continue
        #remaining case: fbm>fm and fam>fm
        assert (fbm > fm) and (fam > fm)
        xa, fa = xam, fam
        xb, fb = xbm, fbm
        assert fm == fm_old
    print "bracket:", xm, fm
    assert fm==f(xm), "{} vs {} vs {}".format(fm_old,fm,f(xm))
    return xm, fm


# vim: tabstop=4 expandtab shiftwidth=4 softtabstop=4 autoindent
```

```
"""File: choose.py
Provides functions for choice of production activity
(determining pclass). Many functions return a dict,
with the following keys:

    pclass: int
        production class (0, 1, 2, 3, 4)
    tr_d : float
        quantity of leisure
    tw_d : float
        time working off-farm
    th_d : float
        quantity of time working on-farm
    ts_d : float
        quantity of time working on-farm
    L_d : float
        quantity of hired labor
    h_d : float
        quantity of land operated
    q_d : float
        output planned.
    Y_d : float
        Income
    U_d : float
        Utility at the maximum
"""
import math, logging
import numpy as np
from scipy.optimize import fminbound

### global constant ###
constraints = dict(
    B=(1<<0),
    R1=(1<<1),
    rA=(1<<2),
    ell0=(1<<3),
    )



def cultivate(agent, K, v, w, rho, params):
    """Return dict of agent attributes,
    including pclass, factor demands, and expected welfare.

    Parameters
    ----------
    agent : Producer
        (Need its hbar and bbar.)
    K : float
        sector specific fixed cost (KT or KM)
    v : float
        Price of land
    w : float
        Price of labor
    rho : float
        Premium associated with participation in modern value chain
    params : dict
        Model parameters, especially s1, s2, D, KM, KT.
        (See distribution.py for the definitions)

    Returns
    -------
    agentinfo : dict
        keys are pclass,tr_d,tw_d,th_d,L_d,h_d,q_d,Y_d,lmda,U_d
        (See the module documentation for definitions.)
    """
    assert K in (params['KT'],params['KM']), 'unrecognized fixed cost type'
    assert rho >= 1, "rho={}; caller shd ensure rho >= 1".format(rho)
```

```
    B = agent.bbar + v*agent.hbar - K  #working capital depends on hbar and K
    if math.isnan(B):
        msg = "B={}, bbar={}, hbar={}, K={}, v={}, w={}, rho={}"
        msg = msg.format(B,agent.bbar,agent.hbar,K,v,w,rho)
        raise ValueError(msg)
    B1, B2, B3 = b123(agent, K, v, w, rho, params)
    if B < B1:
        # Producer is LC (laborer_cultivator )
        agentinfo = laborer_cultivator(agent, B, K, v, w, rho, params)
    else:
        # Producer is SC, SM, or LG
        agentinfo = cultivator(agent, B, K, v, w, rho, params)
    agentinfo['K'] = K
    return agentinfo


def b123(agent, K, v, w, rho, params):
    """Return 3-tuple of float,
    the values of B1, B2, and B3.
    """
    assert (K==params['KT'] or K==params['KM'])
    A = agent.A        #Productivity parameter
    s1 = params['s1'] #first-order aspect of supervision function
    s2 = params['s2'] #second-order aspect of supervision function
    D = params['D']    #scale factor: sub-utility of leisure
    dar = D / (rho * A)
    dar2 = dar * dar
    dar4 = dar2 * dar2
    B1 = w - v*dar*dar
    B2 = B1 + w* s1/(1 - s1)
    a2r2 = A*A*rho*rho
    assert np.allclose(B2, w/(1 - s1) - v*dar*dar)
    old_disc = (4*s2**2*D**4*v**2 - 4*a2r2*s2*D**2*v*w + a2r2*a2r2*(s1**2 + 4*s2)*w**2)
    oldB3 = -(2*s1*s2*D**2*v - 2*s1**2*s2*D**2*v - 8*s2**2*D**2*v \
        + a2r2*(-2*s1*w + s1**2*w + s1**3*w + 4*s2*w + 4*s1*s2*w) \
        - (-2 + s1 + s1**2 + 4*s2) \
        * np.sqrt(4*s2**2*D**4*v**2 - 4*a2r2*s2*D**2*v*w + a2r2*a2r2*(s1**2 + 4*s2)*w**2))\
        /(2*a2r2*s2*(-1 + s1**2 + 4*s2))
    disc = (s2*s2*dar4*v*v - s2*dar2*v*w + (s1**2/4 + s2)*w*w)
    assert np.allclose(old_disc, 4*a2r2*a2r2 * disc)
    if disc < 0: #shd only happen due to rounding error
        assert np.allclose(0, disc), "{}".format(disc)
        disc = 0
    B3 = ((s1 - s1*s1*(1 + s1)/2 - 2*s2 - 2*s1*s2)*w \
        + (s1 + s1**2 + 4*s2 - 2) * np.sqrt(disc) \
        - dar2*v*s2*(s1 - s1**2 - 4*s2)) \
       /(s2*(-1 + s1**2 + 4*s2))
    if disc > 0:
        assert np.allclose(B3,oldB3)
    assert B1 <= B2 <= B3, 'B1:{B1}, B2:{B2}, B3:{B3}'.format(B1=B1,B2=B2,B3=B3)
    return B1, B2, B3


def cultivator(agent, B, K, v, w, rho, params):
    """Return dict of producer attributes,
    including pclass, factor demands, and welfare.
    1. Try SM. If th<0, switch to LG. Elif L<0 switch to SC.
    2. If capital constraint binds, delegate to constrained_cultivator.

    Parameters
    ----------
    agent : Producer
        the agent (we need its hbar and A attributes)
    K : float
        Fixed set-up costs
    rho : float
        Premium associated with participation in modern value chain
    v : float
```

```
    Price of land
w : float
    Price of labor
params : dict
    Model parameters, especially s1, s2, D, KM, KT.
    (see distribution.py for the definitions)

Returns
-------
result : dict
    keys are pclass,tr_d,tw_d,th_d,L_d,h_d,q_d,Y_d,lmda,U_d
    (See the module documentation for definitions.)

Warning:
--------
Currently has multiple (3) return points.
"""
assert (K==params['KT'] or K==params['KM'])
hbar, A = agent.hbar, agent.A
s1, s2, D = params['s1'], params['s2'], params['D']
result = dict(tw_d=0, constraints=0)
dar = D / (A * rho)
dd2  = D*D
aa2 = A*A
rr2 = rho*rho
vv2 = v*v
a2r2 = aa2*rr2
ar4vw01 = a2r2 - 4*v*w
ar4vw02 = a2r2 / (4 * v * w)
#compute SMU01
pclass = 3   #SM
L_u = (1-s1-1/ar4vw02)/(2*s2)
#
assert np.allclose(L_u, (1-s1)/(2*s2) - (2 * v * w)/(s2*a2r2) )
#
tr_u = (4*dar*dar*vv2)/a2r2
ts_u = s1*L_u + s2*L_u*L_u
th_u = 1 - tr_u - ts_u
h_u = (w/v)*ar4vw02*(1+L_u-ts_u-tr_u)
if (L_u < 0): #try SCU instead (i.e., impose L=0)
    assert (ar4vw02 < 1/(1-s1))
    if (a2r2 <= 2 * D * v): #production does not pay
        result = pure_laborer(agent, v, w, params)
        result['Y_d'] -= K
        result['U_d'] -= K
        return result
        #^^^^^ #:todo: ugly return point
    else: #SCU
        pclass = 2
        L_u = 0
        ts_u = 0
        h_u = a2r2/(4*vv2) - dar*dar
        tr_u = 4*v*v*dar*dar/a2r2
        th_u = 1 - tr_u
        assert np.allclose(th_u, h_u*4*vv2/a2r2)
elif (th_u < 0): #try LGU instead
    assert (ar4vw01 >= 2*D*v*s1)  #weighted average of above conditions -> L>=0 for LGU
    pclass = 4
    th_u = 0
    s12 = s1/(2*s2)
    L_u = -s12 + math.sqrt(s12*s12 + 1/s2) * ar4vw01 / math.sqrt(16*dd2*vv2*s2+ar4vw01*ar4vw01)
    ts_u = s1*L_u + s2*L_u*L_u
    assert L_u>=0, 'L_u={} (should have L_u>0 if it pays to produce)'.format(L_u)
    h_u = (a2r2/vv2) * (L_u / 4)
    tr_u = 1 - ts_u
    assert tr_u >= 0, 'L_u={0} should be constrained to {1}'.format(L_u, maxL)
else: #SMU applies
    assert np.allclose(h_u, (w/v)*(1+(1-s1)*L_u-s2*L_u*L_u)/(1-s1-2*s2*L_u)-dar*dar),\
```

```
        "{} vs. {}".format(h_u,  (w/v)*(1+(1-s1)*L_u-s2*L_u*L_u)/(1-s1-2*s2*L_u)-dar*dar)
    #test for capital constraint
    if (v*h_u + w*L_u <= B): #does not bind! we're done!
        assert (0 <= tr_u <= 1)
        msg = "cultivator: for pclass {}, capital constraint does not bind".format(pclass)
        logging.debug(msg)
        q_u = A * math.sqrt(h_u*L_u)
        Y_u = rho*q_u - v*(h_u - hbar) - w*L_u - K
        U_u = Y_u + D*math.sqrt(tr_u)
        result.update(pclass=pclass
                      ,K=K
                      ,h_d = h_u
                      ,L_d=L_u
                      ,tr_d = tr_u
                      ,th_d=th_u
                      ,q_d=q_u
                      ,Y_d=Y_u
                      ,U_d=U_u
                      ,lmda=0
                      )
    else: #capital constraint binds; fetch constrained result
        if (pclass == 4):
            assert L_u > 4*B*v / (a2r2 + 4*v*w), 'LG: L_u should be greater than max unconstrained L'
        msg = "cultivator: for pclass {}, capital constraint binds".format(pclass)
        logging.debug(msg)
        result = constrained_cultivator(agent, B, K, v, w, rho, params) #switch to constrained result
    return result

def constrained_cultivator(agent, B, K, v, w, rho, params):
    """Return dict of agent attributes,
    including pclass, factor demands, and welfare.
    """
    assert B==agent.bbar + v*agent.hbar - K
    B1, B2, B3 = b123(agent, K, v, w, rho, params)
    assert (B >= B1)
    if B1 <= B <= B2:  # Producer is self_cultivator
        agentinfo = self_cultivator(agent, B, K, v, w, rho, params)
    elif B2 < B < B3:    # Producer is small capitalist
        agentinfo = small_capitalist(agent, B, K, v, w, rho, params)
    elif (B >= B3):                # Producer is large capitalist
        agentinfo = large_capitalist(agent, B, K, v, w, rho, params)
    else:
        msg = "Bad B={} (vs B1={}, B2={}, B3={}".format(B,B1,B2,B3)
        raise ValueError(msg)
    return agentinfo


def large_capitalist(agent, B, K, v, w, rho, params):
    """Return dict of agent attributes,
    including pclass, factor demands, and welfare.

    Parameters
    ----------
    B : float
        exogenous working capital (may be negative due to K)
    hbar : float
        Quantity of land owned
    K : float
        Fixed set-up costs
    rho : float
        Premium associated with participation in modern value chain
    v : float
        Price of land
    w : float
        Price of labor
    params : dict
        Model parameters, especially s1, s2, D, KM, KT.
        (see distribution.py for the definitions)
```

```
Returns
-------
result : dict
    keys are pclass, tr_d, tw_d, th_d, L_d, h_d, q_d, Y_d, lmda, U_d
    (See the module documentation for definitions.)
"""
assert (K==params['KT'] or K==params['KM'])
result = dict(pclass=4, tw_d=0, th_d=0, constraints=0)
B1, B2, B3 = b123(agent, K, v, w, rho, params)
assert (B >= B3)
hbar = agent.hbar
A = agent.A #Productivity parameter on Cobb-Douglas production function
s1 = params['s1']
s2 = params['s2']
D = params['D']
#1. check that it pays LG to produce
dar = D / (A * rho)
aa2 = A*A
rr2 = rho*rho
a2r2 = aa2*rr2
dd2  = D*D
vv2 = v*v
ar4vw01 = a2r2 - 4*v*w
s12 = s1/(2*s2)
# compute constrained solution
# **tentative** constrained
ar4vw02 = a2r2 / (4 * v * w)
minL = (B/w) / (1 + ar4vw02)
maxL = (math.sqrt(s1*s1+4*s2)-s1)/(2*s2)  #highest possible L value (asymptote)
hmax = (B/v) * ar4vw02 / (1 + ar4vw02)
####old appraoch
ww2 = w*w
v04 = B*B
v05 = 1.0/w
v06 = 1.0/s2
v07 = s1*w
v08 = 1.0 / ww2
s2s2i = 1.0 / (s2*s2)
v10 = -(B*s2)
v11 = 2**(-1.0/3)
v12 = 2**(-2.0/3)
v13 = s2*v*dd2
v14 = s1**2*v*dd2
v15 = v07 + v10
v16 = 4*s1*B*v13
v17 = v14*w
v18 = -v17
v19 = v15**2
v20 = -4*ww2*a2r2
v21 = -4*B*a2r2*v07
v22 = s2*a2r2*v04
v23 = v13 - a2r2*w
v24 = 1.0/v23
v25 = s1*B*a2r2 + v14 + 4*a2r2*w
v26 = s1*ww2*a2r2 + B*s2**2*v*dd2 - v07*v13 + a2r2*v10*w
v27 = -108*s2*v04*v23*v25**2*w
v28 = v16 + v18 + v20 + v21 + v22
v29 = 4*ww2*a2r2 + 4*B*a2r2*v07 - 4*s1*B*v13 + v17 - v22
v30 = -432*a2r2*v04*v26**2
v31 = (2*B*a2r2*v07 - 2*s1*B*v13 + v18 + v20 + v22)**2
v32 = 2*v28**3
v33 = -(v05*v06*v24*v29)/4.0
v34 = -288*(s2*a2r2*v04 + v16 + v18 + v20 + v21)*v22*v23*w
v35 = -36*B*v25*v26*v28
v36 = -4*(-12 *B*v25*v26 + v28**2 + 48*v22*v23*w)**3
v37 = v27 + v30 + v32 + v34 + v35
v38 = np.sqrt(v36 + v37**2)
```

```
v39 = np.sqrt((v27 + v30 + v32 + v34 + v35)**2 + v36) + v37
v40 = v39**(1/float(3))
v41 = v39**(-1/float(3))
v42 = (v08*s2s2i*v19)/4.0 - (v05*v06*v24*v28)/12.0 + v33 - \
      (v05*v06*v11*v24*v40)/float(12) - (v05*v06*v12*v24*v31*v41)/6.0
L_c_old = (v05*v06*(3*B*s2 - 3*v07- 6*s2*np.sqrt(v42)*w +
    6*s2*np.sqrt((v08*s2s2i*v19)/float(2) +
    (v05*v06*v24*v28)/float(12) + v33 +
    (v05*v06*v11*v24*v40)/float(12) +
    (v05*v06*v12*v24*v31*v41)/float(6) -
    (2*B*v05*v06*v24*v25 + v08*s2s2i*v15*v24*v29 -
    v15**3/(s2**3*w**3))/(4*np.sqrt(v42)))*w))/float(12)
### new approach
#notation
bbw = B/w
bw2 = bbw * bbw
bw3 = bw2 * bbw
bw4 = bw2 * bw2
q0 = math.sqrt(w/v)/dar
q2 = q0 * q0
q4 = q2 * q2
q6 = q4 * q2
s12bw = s1/s2 - bbw
#L1
L1 = -(1./4.)*s12bw
#L2
L2c1 = 2*(-(s1**3*(s1 + 2*bbw*s2)**3) + q6*(-4 + 2*bbw*s1 + bw2*s2)**3
       + 3*q4*(2*s1**2*(-8 + bbw*s1*(8 + 7*bbw*s1))
       + 8*bbw*s1*(-4 + bbw*s1*(14 + 3*bbw*s1))*s2
       + bw3*s1*(160 + 9*bbw*s1)*s2**2 - 2*bw4*(-36 + bbw*s1)*s2**3)
       + 3*q2*(2*s1**4*(-2 + bbw*s1) - bbw*s1**3*(16 + 9*bbw*s1)*s2
       - 8*bw2*s1**2*(11 + 3*bbw*s1)*s2**2
       - 2*bw3*s1*(72 + 7*bbw*s1)*s2**3 - 72*bw4*s2**4))
#L2c2i = -4 * (s1*(s1 + 2*bbw*s2) - q2*(-4 + 2*bbw*s1 + bw2*s2))**6
sqrtnegL2c2i = 2 * (s1*(s1 + 2*bbw*s2) - q2*(-4 + 2*bbw*s1 + bw2*s2))**3
#print "L2c2i,L2c1,L2c2i + L2c1*L2c1", L2c2i, L2c1, L2c2i + L2c1*L2c1
try:
    #L2c2 = math.sqrt(L2c2i + L2c1*L2c1) #can fail due to rounding error or overflow
    L2c2 = math.sqrt((L2c1+sqrtnegL2c2i)*(L2c1-sqrtnegL2c2i)) #can fail due to rounding error
except ValueError:
    msg = "L2C2 ValueError: v={}, w={}, rho={}: ".format(v, w, rho)
    logging.info(msg)
    print(msg)
    print(L2c1,sqrtnegL2c2i)
    assert np.allclose(L2c1,abs(sqrtnegL2c2i))
    L2c2 = 0
L2c = L2c1 + L2c2
L2a = 8*q2 + 8*bbw*q2*s1 + 2*s1*s1 - 2*bw2*q2*s2 - 8*bbw*s1*s2
L2b = (-4*q2 + 2*bbw*q2*s1 - s1*s1 + bw2*q2*s2 - 2*bbw*s1*s2)**2
pt1 = (1./4.)*s12bw**2 + L2a/(12*(q2 - s2)*s2)
pt2 = (1./(12*(q2 - s2)*s2))*(2**(1./3.)*L2b/L2c**(1./3.) + 2**(-1./3.)*L2c**(1./3.))
L2 = -(1./2.)*math.sqrt(pt1 + pt2)
#L3
L3a = L2a/(-2*(q2 - s2)*s2)
pt3 = (-2*bbw*(4*q2 + bbw*q2*s1 + s1*s1)/((q2 - s2)*s2) - s12bw**3 + s12bw*L3a)/(8*L2)
discL3 = 2*pt1 - pt2 + pt3
if discL3 < 0:  #should only happen via rounding error
    print("problem with discL3 {}".format(discL3))
    #:todo: replace the following, which can fail numerically because of subtractve cancellatio;
    #add a better test
    #assert np.allclose(discL3, 0), "{}".format(discL3)
    discL3 = 0
L3 = (1./2.)*math.sqrt(discL3)
#old version (kept for reference):
#L3 = (1./2.)*math.sqrt(2*pt1 - pt2 - (-(( 2*bbw*(4*q2 + bbw*q2*s1 + s1*s1))
#                              /((q2 - s2)*s2)) - s12bw**3 + s12bw*L3a)/(-8*L2))
L_c = L1 + L2 + L3  #constrained optimum
if math.isnan(L_c) or (v<1e-8 or w<1e-8) or __debug__:
```

```
        obj2min = lambda L: B + K - D*math.sqrt(1-s1*L-s2*L*L) - v*hbar - A*rho*math.sqrt(L*(B-w*L)/v)
        numL = fminbound(obj2min, 0, (math.sqrt(s1*s1+4*s2)-s1)/(2*s2))
        if numL > maxL:
            msg = "\neksupp2 = {{v->{v}, w->{w}, \\[Rho]->{rho}, hbar->{hbar}, k->{k}, bb->{bb}}}"
            msg = msg.format(v=v,w=w,rho=rho,hbar=hbar,k=K,bb=B)
            msg += "should have numL={} < maxL={}".format(numL,maxL)
            logging.warn(msg)
            assert np.allclose(numL, maxL)
            numL = maxL - 1e-12   #:note: HACK! (just worrying that bound not tightly enforced)
    if math.isnan(L_c) or (v<1e-8 or w<1e-8):
        L_c = numL
        msg = "new L_c calculation failed; use numL"
        msg += "({numL})".format(numL=numL)
        print msg
        logging.info(msg)
    elif __debug__:
        assert np.allclose(L_c, numL), "{} vs {}+{}+{}".format(numL, L1, L2, L3) + msg
        if not math.isnan(L_c_old):
            assert np.allclose(L_c, L_c_old), "{} vs {}+{}+{}".format(L_c_old, L1, L2, L3)
            assert np.allclose(L_c_old,numL)
        else:
            msg = "old L_c calculation failed"
            logging.info(msg)
            print 'v19:{v19}, v39:{v39}, v42:{v42}, B:{B}'.format(v19=v19,v39=v39, v42=v42,B=B)
            msg = "L_c_old={L},numL={numL}".format(L=L_c_old,numL=numL)
            print msg
            logging.info(msg)
    assert L_c > minL, "constrained solution {} should exceed {}.".format(L_c,minL)
    h_d = (B - w*L_c)/v
    if h_d > hmax: #should only happen from rounding error
        assert np.allclose(hmax/h_d,1.0), "{} vs. {}".format(h_d, hmax)
        h_d = hmax
    #:note: set implied Lagrange multiplier
    result['lmda'] = (A*L_c**(0.5)*rho)/(2*h_d**(0.5)*v) - 1
    assert L_c>=0, 'L_c={} should be nonnegative'.format(L_c)
    ts_c = s1*L_c + s2*L_c*L_c
    if ts_c > 1: #should only happen due to rounding error
        assert np.allclose(L_c,maxL)
        msg = 'LG: L_c={0}; maxL={1}'.format(L_c,maxL)
        logging.warn(msg)
        print(msg)
        assert np.allclose(ts_c, 1.0)
        L_c = maxL - 1e-12   #:note: HACK! but o/w rounding error can cause ts>1
        ts_c = s1*L_c + s2*L_c*L_c
        assert ts_c < 1
    assert s1*L_c + s2*L_c*L_c <= 1, 'L_c={0} yields ts={1}'.format(L_c,s1*L_c + s2*L_c*L_c)
    assert h_d>=0, 'h_d={}'.format(h_d)
    result['L_d'] = L_d = L_c
    result['h_d'] = h_d
    result['tr_d'] = tr_d = 1 - s1*L_d - s2*L_d*L_d
    assert tr_d>=0, 'tr_d={}'.format(tr_d)
    result['q_d'] = q_d = A * math.sqrt(h_d*L_d)
    result['Y_d'] = Y_d = rho*q_d - v*(h_d - hbar) - w*L_d - K
    result['U_d'] = Y_d + D*math.sqrt(tr_d)
    return result

def small_capitalist(agent, B, K, v, w, rho, params):
    """Return dict of agent attributes,
    including pclass, factor demands, and welfare.

    Parameters
    ----------
    B : float
        exogenous working capital (may be negative due to K)
    hbar : float
        Quantity of land owned
    K : float
        Fixed set-up costs (equals KT or KM)
```

```
rho : float
    Premium associated with participation in modern value chain
v : float
    Price of land
w : float
    Price of labor
params : dict
    Model parameters, especially s1, s2, D, KM, KT.
    (see distribution.py for the definitions)

Returns
-------
result : dict
    keys are pclass,tr_d,tw_d,th_d,L_d,h_d,q_d,Y_d,lmda,U_d
    (See the module documentation for definitions.)
"""
assert (K==params['KT'] or K==params['KM'])
#TODO: check that it pays to produce at all (rho must be high enough)
hbar, A = agent.hbar, agent.A
s1, s2, D = params['s1'], params['s2'], params['D']
result = dict(pclass=3, tw_d=0, constraints=0)
#NEW approach
#1. FIRST: check that it is profitable to hire labor
dar = D / (A * rho)
aa2 = A*A
rr2 = rho*rho
a2r2 = aa2*rr2
ar4vw02 = a2r2 / (4 * v * w)

#original formulation
R_c = (D**2*v*(-2*s2*D**2*v + A**2*rho**2*(-2*B*s2 + w - s1*w) +
    2*np.sqrt(s2**2*(A**2*B*rho**2 + D**2*v)**2 +
    A**2*(-1 + s1)*s2*rho**2*(A**2*B*rho**2 + D**2*v)*w +
    A**4*((-1 + s1)**2 + 3*s2)*rho**4*w**2)))/(3*A**4*rho**4*w**2)
Ld_c = (s2*D**2*v + A**2*rho**2*(B*s2 + w - s1*w) -
    np.sqrt(s2**2*(A**2*B*rho**2 + D**2*v)**2 +
    A**2*(-1 + s1)*s2*rho**2*(A**2*B*rho**2 + D**2*v)*w +
    A**4*((-1 + s1)**2 + 3*s2)*rho**4*w**2))/(3*A**2*s2*rho**2*w)

#preferred formulation
x = (B + dar*dar*v) / w
L_d = (1-s1+s2*x - math.sqrt((1-s1-x*s2)**2+3*s2+(1-s1)*x*s2))/(3*s2)
tr_d = (1-s1-2*s2*L_d)*dar*dar*v/w


#yet another (Mma based) formulation
L201506 = (D**2*s2*v
         + A**2*rho**2*(B*s2 + w - s1*w)
         - math.sqrt(3*A**2*rho**2*s2*w* (D**2*(-1 + s1)*v + A**2*rho**2*(B*(-1 + s1) + w))
                + (D**2*s2*v + A**2*rho**2*(B*s2 + w - s1*w))**2)
        ) / (3.*A**2*rho**2*s2*w)
tr201506 = (D**2*v*(
                    -2*D**2*s2*v
                    + A**2*rho**2*(-2*B*s2 + w - s1*w)
                    + 2*math.sqrt(3*A**2*rho**2*s2*w
                                  * (D**2*(-1 + s1)*v + A**2*rho**2*(B*(-1 + s1) + w))
                                  + (D**2*s2*v + A**2*rho**2*(B*s2 + w - s1*w))**2
                                 )
                   )
           ) / (3.*A**4*rho**4*w**2)

#these are complicated enough to check various expressions against each other:
assert np.allclose(L201506, Ld_c)
assert np.allclose(L_d, Ld_c)
assert np.allclose(tr201506, R_c)
assert np.allclose(tr_d, R_c)

h_d = (B - w*L_d) / v
```

```
    assert B< ((1 - s1)*w)/(2*s2) + (A*A*(1-s1)*(1-s1)*rho*rho)/(16*s2*v) - (3*v*w*w)/(A*A*s2*rho*rho) \
        + (rho*rho*A*A*(1-tr_d))/(4*v), "see solutions"  #:note: *unconstrained* tr_d here
    #TODO: Simplify lmda?
    lmda = np.sqrt(v*w**2*(-2*s2*D**2*v + A**2*rho**2*(-2*B*s2 + w - s1*w) +
       2*np.sqrt(s2**2*(A**2*B*rho**2 + D**2*v)**2 +
       A**2*(-1 + s1)*s2*rho**2*(A**2*B*rho**2 + D**2*v)*w +
       A**4*((-1 + s1)**2 + 3*s2)*rho**4*w**2)))/(2*np.sqrt(3)*v*w**2) - 1
    if L_d < 0:
        msg = "L={L}".format(L=L_d)
        logging.warn(msg)
        print "LG:", msg
        L_d = 0
    if tr_d < 0:
        msg = "tr_d={tr_d}".format(tr_d=tr_d)
        logging.warn(msg)
        print "LG:", msg
        tr_d = 0
    if tr_d > 1:
        result['constraints'] |= constraints['R1']
        logging.warn('LG: time constraint limits tr_d to 1.')
        print('LG: time constraint limits tr_d to 1.')
        tr_d = 1
    if h_d < 0:
        msg = "h={h}".format(h=h_d)
        logging.warn(msg)
        print "LG:", msg
        h_d = 0
    ts_d =  s1*L_d + s2*L_d*L_d
    th_d = 1 - tr_d - ts_d
    if th_d < 0:
        msg = "th_d={th_d}".format(th_d=th_d)
        logging.warn(msg)
        print "LG:", msg
        th_d = 0
    result['L_d'] = L_d
    assert L_d>=0, 'L_d={}'.format(L_d)
    result['tr_d'] = tr_d
    assert 1>=tr_d>=0, 'tr_d={}'.format(tr_d)
    result['h_d'] = h_d
    assert h_d>=0, 'h_d={}'.format(h_d)
    result['th_d'] = th_d
    assert 1>=th_d>=0, 'th_d={}'.format(th_d)
    result['ts_d'] = ts_d
    assert 1>=ts_d>=0, 'ts_d={}'.format(th_d)
    result['q_d'] = q_d = A * math.sqrt(h_d*(th_d+L_d))
    result['Y_d'] = Y_d = rho*q_d - v*(h_d-hbar) - w*L_d - K
    result['U_d'] = Y_d + D*math.sqrt(tr_d)
    result['lmda'] = lmda
    return result

def self_cultivator(agent, B, K, v, w, rho, params):
    """Return dict of agent attributes,
    including pclass, factor demands, and welfare.

    Parameters
    ----------
    hbar : float
        Quantity of land owned
    B : float
        exogenous working capital (may be negative due to K)
    K : float
        Fixed set-up costs (equals KT or KM)
    rho : float
        Premium associated with participation in modern value chain
    v : float
        Price of land
    w : float
        Price of labor
```

```
    params : dict
        Model parameters, especially s1, s2, D, KM, KT.
        Here we need A and D.
        (See distribution.py for the definitions)

    Returns
    -------
    result : dict
        keys are pclass,tr_d,tw_d,th_d,L_d,h_d,q_d,Y_d,lmda,U_d
        (See the module documentation for definitions.)
    """
    assert (K==params['KT'] or K==params['KM'])
    assert B>0, "if B<=0, cannot cultivate"
    assert v>0, "if v<=0, problem ill defined; solutions nonsense"
    hbar = agent.hbar
    A = agent.A #Productivity parameter on Cobb-Douglas production function
    D = params['D']
    dar = D / (rho * A)
    result = dict(pclass=2, tw_d=0, L_d=0, constraints=0)
    #CONSTRAINED and UNCONSTRAINED SOLUTIONS
    h_c = B/v
    h_u = (A*A*rho*rho)/(4*v*v) - dar*dar #unconstrained
    assert (h_u > 0)
    if h_u <= 0: #does not pay to produce, delegate to PAL
        result.update(pure_laborer(agent, v, w, params)) #sets pclass=0
        result['U_d'] -= K
        result['lmda'] = 0
        #result['constraints'] |= constraints['R1']
        logging.warn('self_cultivator: delegate to pal')
    # note: th_d->0 as h->0 (in both cases)
    assert (h_c < h_u)  #usual case: working capital constraint binds
    assert 0 < B < (rho*rho*A*A/(4*v) - dar*dar*v), "violated upper bound on B for constraint to bind"
    result['constraints'] |= constraints['B']
    h_d = h_c
    th_d = h_c/(dar*dar+h_c) #constrained value
    assert np.allclose(th_d, A*A*h_d*rho*rho/(D*D+A*A*h_d*rho*rho)) #0 < th_d < 1
    lmda = A*rho/(2 *v*math.sqrt(dar*dar+B/v)) - 1
    lmda02 = (A**2*rho**2)/(2*v**(0.5)*(A**2*B*rho**2 + v*D**2)**(0.5)) - 1
    lmda03 = 0.5*A*rho*math.sqrt(th_d/(v*B)) - 1
    if not np.allclose(lmda, lmda02, lmda03):
        print "sign problem? {} vs {} vs {}".format(lmda,lmda02, lmda03)
        print A, rho, v, B
        raise ValueError()
    #record SC solutions
    if result['pclass'] == 2: #o.w. delegated to PAL
        result['h_d'] = h_d
        result['th_d'] = th_d
        result['tr_d'] = tr_d = 1 - th_d   #1 if h=0
        result['q_d'] = q_d = A * math.sqrt(h_d*th_d)  #0 if h=0
        result['Y_d'] = Y_d = rho*q_d + v*(hbar - h_d) - K
        if h_c > h_u:
            assert np.allclose(Y_d, th_d*A*A*rho*rho/(4*v) + v*hbar - K)
        result['U_d'] = Y_d + D*math.sqrt(tr_d)
        result['lmda'] = lmda
    else:
        assert result['pclass'] == 0, "only delegation is to PAL"
    return result

def laborer_cultivator(agent, B, K, v, w, rho, params):
    """Return dict of agent attributes,
    including pclass, factor demands, and welfare.

    Parameters
    ----------
    hbar : float
        Quantity of land owned
    B : float
        exogenous working capital (may be negative due to K)
```

```
    K : float
        Fixed set-up costs
    rho : float
        Premium associated with participation in modern value chain
    v : float
        Price of land
    w : float
        Price of labor
    params : dict
        Model parameters, especially s1, s2, D, KM, KT.
        Here we need A and D.
        (See distribution.py for the definitions)

    Returns
    -------
    result : dict, laborer_cultivator's attributes,
        keys are pclass,tr_d,tw_d,th_d,L_d,h_d,q_d,Y_d,lmda,U_d
        (See the module documentation for definitions.)
"""
    assert (K==params['KT'] or K==params['KM'])
    B1, B2, B3 = b123(agent, K, v, w, rho, params)
    #B < B1 = w(1-tr_d) here
    assert B<B1, "caller should enforce B<B1"
    hbar = agent.hbar
    A = agent.A #Productivity parameter on Cobb-Douglas production function
    D = params['D']
    dar = D / (rho * A)
    result = dict(pclass=1, L_d=0, constraints=0)
    #if it pays to produce then capital constraint binds, otherwise ...
    if (rho * A <= 2 * math.sqrt(v*w)) or (B+B1<=0): #doesn't pay to cultivate (see below)
        result.update(pure_laborer(agent, v, w, params))
        #fixed costs already incurred
        result['U_d'] -= K
        result['Y_d'] -= K
        result['lmda'] = 0   #:note: perhaps None wd be more informative?
        if (rho * A <= 2 * math.sqrt(v*w)): #this happens a lot!
            result['constraints'] |= constraints['rA']
            msg = "LC: rho*A too low to cultivate; delegate to PL."
            logging.debug(msg)
        if (B+B1<=0): #happens *very* often (at least out of eq)
            result['constraints'] |= constraints['ell0']
            msg = "LC: can't afford land to cultivate; delegate to PL."
            logging.debug(msg)
    else:  #:note: if it pays to be an LC, working capital constraint binds
        tr_d, tw_d, th_d, h_d = 1 - B1/w, (B1-B)/(2*w),  (B1+B)/(2*w), (B1+B)/(2*v)
        #as -B1 < B < B1, all vars are properly bounded
        assert np.allclose([tr_d,tw_d,th_d,h_d],[(v/w)*dar*dar,(1-tr_d-B/w)/2,(1-tr_d+B/w)/2,(w/v)*th_d])
        result['tr_d'] = tr_d
        result['tw_d'] = tw_d
        result['th_d'] = th_d
        result['h_d'] = h_d
        assert np.allclose(v*h_d, w*th_d) #equal expenditure on each factor
        result['q_d'] = q_d = A * math.sqrt(h_d*th_d)
        result['Y_d'] = Y_d = rho*q_d + w*tw_d + v*(hbar-h_d) - K
        assert np.allclose(q_d, A*(1 - tr_d + B/w)*math.sqrt(w/v)/2.,  A*(B1 + B)/math.sqrt(4*w*v))
        assert np.allclose(Y_d, 0.5*A*rho*(B+w*(1-tr_d))/math.sqrt(v*w) + v*hbar - B- K)
        result['U_d'] = Y_d + D*math.sqrt(tr_d)
        result['lmda'] = A*rho/(2*math.sqrt(v*w))-1
    return result

def pure_laborer(agent, v, w, params):
    """Return dict of agent attributes,
    including pclass, factor demands, and welfare.

    Parameters
    ----------
    agent : Producer or AgtInfo
      we need access to the hbar attribute
```

```
      (quantity of land owned)
    v : float
      price of land
    w : float
      price of labor
    params : dict
      model parameters;
      we just need access to D.
      (see baseline.ini for the definitions)

    Returns
    -------
    result : dict, pure_laborer's attributes,
        keys are pclass,tr_d,tw_d,th_d,L_d,h_d,q_d,Y_d,lmda,U_d
        (See the module documentation for definitions.)
    """
    assert (v > 0 and w > 0),\
        "negative factor prices: v={}, w={}".format(v,w)
    result = dict(pclass=0
                  ,K=0
                  ,th_d=0
                  ,ts_d=0
                  ,L_d=0
                  ,h_d=0
                  ,q_d=0
                  ,constraints=0
                  )
    D = params['D'] #Parameter on sub-utility (of leisure) function
    d2w = D / (2.0 * w)
    #optimum to pure laborer's objective, subject to tr_d<=1
    if (d2w < 1.0):
        tr_d = d2w * d2w
    else:
        tr_d = 1.0
        result['constraints'] |= constraints['R1']
    result['tr_d'] = tr_d
    result['tw_d'] = tw_d = 1 - tr_d     #PL has only two uses of time
    result['q_d'] = 0.0
    result['Y_d'] = Y_d = w*tw_d + v*agent.hbar #no production by PL
    Utr = D*d2w if (tr_d<1) else Y_d + D #faster than sqrt(tr_d)
    result['U_d'] = Y_d + Utr
    result['lmda'] = None         #capital constraint irrelevant to PL
    return result


# vim: tabstop=4 expandtab shiftwidth=4 softtabstop=4 autoindent
```