HARDPIG

Release 1.0a

Timothy Kam and Ronald Stauber

June 20, 2014

H eterogeneous
    A gent
R ecursive
    D ynamic
P ublic
    I nsurance
G ames.

This web documentation details the methods proposed in the paper by Kam and Stauber, "Computing Dynamic Public Insurance Games with Endogenous Agent Distributions".

`This` is a downloadable PDF version of the documentation.

CHAPTER

ONE

STATE SPACE

Consider the *game state space* as $D := \Delta(\mathcal{Z})$, the set of all probability measures on the finite *individual state-space* $\mathcal{Z} := \{-N, ..., -1, +1, ..., +M\}$, where $0 < N, M < +\infty$. Let $N_z := M + N \equiv |\mathcal{Z}|$.

1.1 Properties

The set D is:

1. is a unit simplex embedded in $\mathbb{R}^{N_z}$:

$$\Delta(\mathcal{Z}) := \left\{ \lambda \in \mathbb{R}^{N_z} : \lambda_i \in [0,1], \forall i = 1, ..., N_z, \text{ and } \sum_{i=1}^{N_z} \lambda_i = 1 \right\}$$

1. represented by a convex polytope (i.e. a unit N_z-simplex);
2. partitioned into $K < +\infty$ equal-area $(N_z - 1)$-simplices, $Q_k, k \in \{1, ..., K\} =: \mathbf{K}$.

Relevant functions ▶

`simplex_tripart`(K)
Returns K number of equal volume simplex partition elements of unit simplex D, given by $Q_k, k = 1, ..., K$.

The next figure–*Example state-space partition scheme* –shows an example where $K = 16$ and $N_z = 3$.

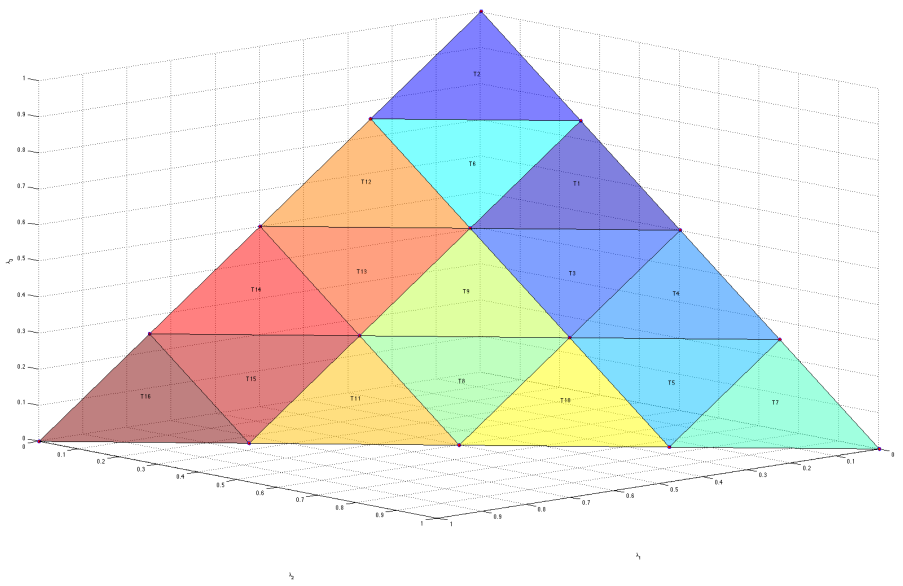

Figure 1.1: Example state-space partition scheme $(N_z = 3, K = 16)$

CHAPTER

TWO

ACTIONS AND TRANSITIONS

Let $\tilde{A} \ni a_j$ denote the finite action set of each individual (small) player $j \in \mathcal{Z}$. Then $A := \tilde{A}^{\mathcal{Z}} \ni a$, denotes the N_z-copies of an individual's action space—i.e. the set of action profiles a. The finite set A has $N_a := |A|$ number of action profiles.

For each action profile $a \in A$, its associated Markov transition probability function is a linear operator $P(a) : D \to D$.

Relevant functions ▶

`A_ProfileSet` (*self*)
: Returns the finite set A of N_a action profiles of individual small players. A numeric array of size $N_a \times N_z$.

`TransProbA` (*self*)
: Returns the finite set of Markov maps $P(a) : D \to D$, one for every $a \in A$. A numeric array of size $N_z \times N_z \times N_a$.

CHAPTER

THREE

INTERSECTIONS WITH STATE-SPACE PARTITIONS

For every $k \in \mathbf{K}$ and its associated simplicial partition element $Q_k \subset D$ with positive volume, the set-valued image $P(a)(Q_k)$:

1. is another N_z-simplex contained in the unit N_z-simplex D; and
2. intersects with:
   - at least one partition element $Q_{k'}$ where $k' \in \mathbf{K}$ and
   - at most all partition elements $Q_1, ..., Q_K$;

3.1 Polytope intersection problems

Denote

$$\mathbf{I}(a,k) := \{k' \in \mathbf{K} : P(a)(Q_k) \cap Q_{k'} \neq \emptyset\}, \qquad \forall a \in A, k \in \mathbf{K},$$

as the sets of indexes to respective partition-elements—i.e. $k' \mapsto Q_{k'}$—that contain non-empty intersections with each simplicial image $P(a)(Q_k)$. Each nonempty intersection, induced by each $(a,k) \in A \times \mathbf{K}$ and $P(a)$, is described by

$$Poly_{k'(a,k)} := \{\lambda' \in D : \lambda' \in P(a)(Q_k) \cap Q_{k'} \neq \emptyset \text{ and } k' \in \mathbf{I}(a,k)\}.$$

Note: Each intersection $Poly_{k'}$, for each $k \in \mathbf{K}$ and each $a \in A$, is a *polytope*, and is *at least a simplex*, and is a subset of partition element $Q_{k'}$, where $k' := k'(a,k)$.

These nonempty intersections are such that

$$\bigcup_{k' \in \mathbf{I}(a,k)} Poly_{k'} =: P(a)(Q_k).$$

Example

If $N_z = 3$, then D is a unit 2-simplex, and each $Poly_{k'}$ can be a polygon or a triangular subset in D.

CHAPTER

FOUR

SAMPLING ON STATE-SPACE PARTITIONS

We will be drawing uniform sample vectors of the variable $\lambda \in D$ from each partition element $Q_k, k \in \mathbf{K}$ of the domain D. And then we will also use this in conjunction with the *Polytope intersection problems*.

Each partition element in general is a tetrahedron simplex. We utilize a modified Markov-chain Monte Carlo sampler called the **Hit-and-Run Algorithm**, originally due to [Smi1984]. This algorithm has a desirable property that it can (globally) reach any point in any arbitrarily given bounded set in $\mathbb{R}^n$ in one step. That is, there is a positive probability of sampling from any neighborhood in that set. Moreover, it is proven by [Lov1999] that the Hit-and-Run sampler converges fast (in probability) to a uniform distribution on convex bodies $Q_k \subset \mathbb{R}^n$. [1] [LV2003] note that this algorithm is the fastest in practice.

Hit-and-Run Algorithm

Let $S \subset \mathbb{R}^n$ be a convex region that restricts sample realizations. The aim is to generate sample $X := \{s_m\}_{m=1}^{N_{sim}}$ as a Markov Chain that is (asymptotically) uniformly distributed on S. Define $f(s)$ by *any* continuous and strictly positive probability density function (pdf) on S.

- Start at a given point s in the given set S. Let $m = 1$.
- Propose a new location $s' = s + ld$ by stepping away from s according to a random direction-stepsize pair, (d, l), where the direction d is uniformly distributed on the unit hypersphere $\mathbb{S}^{n-1}$ embedded in $\mathbb{R}^n$; and the stepsize $l \in \mathbb{R}$ is drawn from a *proposal density* $g_m(l|d, s)$.
- Accept proposal move to s' with *acceptance probability* $\tilde{\alpha}_f(s, s')$, or, reject and stay at s (i.e. set $s' = s$) with probability $1 - \tilde{\alpha}_f(s, s')$.
- Then set s_{m+1} as s', and, repeat the procedure again from the first step, and let $m = m + 1$.

To implement this simple algorithm, we need to define the functions $\{g_m(\cdot|d, s) : m \geq 1\}$, and f (which implies $\tilde{\alpha}_f$) to ensure the necessary and sufficient (Kolmogorov) detailed balance condition holds (for the chain to be a *reversible Markov chain*):

$$g_m\left(\|s - s'\|; \frac{s' - s}{\|s - s'\|}, s\right) \tilde{\alpha}_f(s, s') f(s) = g_m\left(\|s - s'\|; \frac{s - s'}{\|s - s'\|}, s'\right) \tilde{\alpha}_f(s', s) f(s').$$

This demands that the products of probabilities around every closed loop are the same in both directions around the loop.

- We can define $f(s)$ by *any* continuous and strictly positive probability density function (pdf) on S.
- Let $L_m := \{l \in \mathbb{R} : s_m + l d_m \in S\}$. Define a conditional *proposal density* for each step $m = 1, ..., N_{sim}$ by $g_m(l|d, s)$.
  - Proposal densities that satisfy the *detailed balance condition* include the class of symmetric proposal density—i.e. $g_m(l|d, s) = \tilde{g}_m(l) = \tilde{g}(-l)$ for all $l \in \mathbb{R}$, in which case

$$\tilde{\alpha}(s, s') = \min\left\{\frac{f(s')}{f(s)}, 1\right\}.$$

[1] [Lov1999] proves that the upper bound on the convergence rate is in polynomial time of $\mathcal{O}(n^3)$.

– Since we have also S bounded, we can define a valid proposal density as

$$\tilde{g}_m(l) = \frac{\mathbf{1}_{\{l \in L_m\}}}{\int_{\mathbb{R}} \mathbf{1}_{\{u \in L_m\}} du}.$$

Example

In our application, we will define a $S := Q_k$ for every $k \in \mathbf{K}$.

See Section 6.3.1 of [KTB2011] for a generalized version of this simple algorithm.

Looking ahead

Here we give a preview of the usage of uniform sampling from the convex partition elements. In *Equilibrium Payoff Correspondence* later, we show that in our class of dynamic games, the description of the *symmetric sequential equilibrium* operator (which is correspondence valued) involves solving many *non-separable* bilinear programs (BLP) of the form:

$$\begin{aligned} \max_{\lambda,w} \quad & u^T + \lambda^T Q w \\ s.t. \quad & \lambda \in \mathcal{F}_1(w), \\ & w \in \mathcal{F}_2(\lambda); \end{aligned}$$

where $u \in \mathbb{R}^{N_z}$ is a vector of constants; $\lambda, w \in \mathbb{R}^{N_z}$ are the variables of interest; Q is some $(N_z \times N_z)$ real matrix; and the constraint sets $\mathcal{F}_1(w)$ and $\mathcal{F}_2(\lambda)$ are convex polytopes which, respectively, depend on the choices of w and λ. [a] [b]

We propose a Monte Carlo or stochastic approach to obtain ϵ-global (i.e. approximately global) optimization solutions to these non-separable bilinear programs. For now, notice that for each given realization of the random vector λ, the nonseparable BLP above can be reduced to standard linear programs (LP) in the variable w. [c]

[a] A special (and textbook case) is where $\mathcal{F}_1(w) \equiv \mathcal{F}_1$ and $\mathcal{F}_2(\lambda) \equiv \mathcal{F}_2$—i.e. each constraint set $\mathcal{F}_1$ and $\mathcal{F}_2$ do not vary, respectively, with the choice variables w and λ. This special case is known as a *separable bilinear* program, and, it nests quadratic programming as another special case. These problems are known to have a global solution–see [BM1993]. Furthermore, successive approximation using branching-and-bounding methods–i.e. branching into subsets of the optimizer domain, then bounding the value function below by the solutions of linear programs on each subset of the function domain, and, above by the value from a local nonlinear optimizer–can be used to find the ϵ-global optimum: [McC1976] , [BM1993] and [HT1996]

[b] In the paper, we noted that in this class of games, the *source of bilinear nonseparability in the constraint sets* of λ and w is the utilitarian government's set of *incentive* or *promise-keeping* constraints.

[c] Note that by fixing each λ, the constraint set $\mathcal{F}_1(w)$ will be redundant in the LP formulation within the stochastic global optimization scheme. Additionally, we will also require each realization λ to be feasible according to some feasibility (e.g. a budget-balance) requirement(s): $\lambda \in \mathcal{F}(Q_k) := \{\lambda \in Q_k : \lambda b^T \geq 0, \forall b \in B\}$, where B is some finite set of action profiles of the large (government) player.

Example (Sampling from Q_k and $Poly_{k'}$)

The following figure (*Uniform samples and various polytope intersections*) shows an example of our usage of the Hit-and-Run algorithm in conjunction with our polytope intersection problems described earlier. For example, consider the (4,1)-panel in this figure. It shows the realizations of the random vectors $\lambda P(a)$, where $a = 4$ denotes the fourth action profile in A, that would end up in the various partition elements of (Q_3, Q_4, Q_5, Q_8), and given that each vector λ is randomly drawn from the set Q_9.

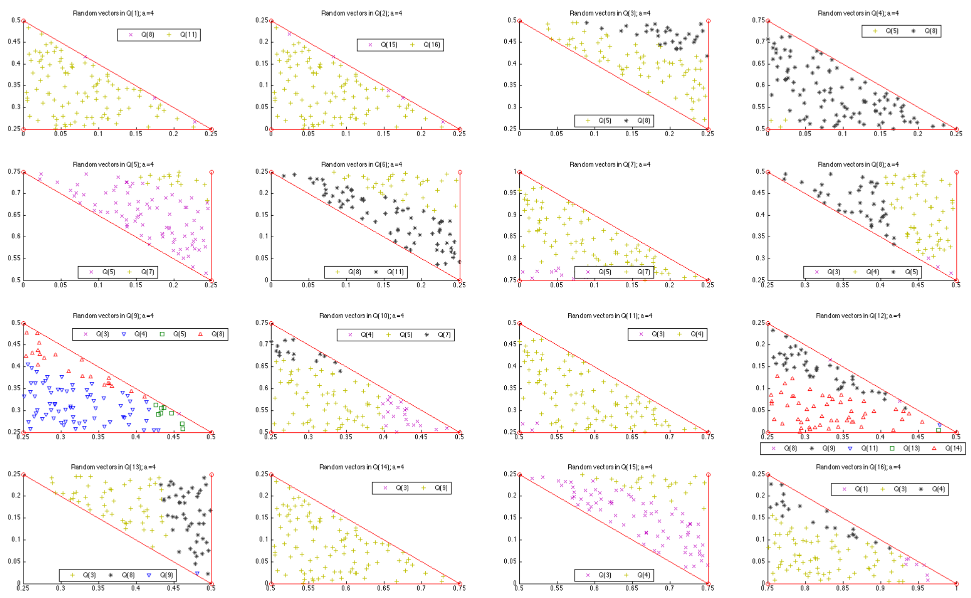

Figure 4.1: Uniform samples and various polytope intersections $(P(a_4), N_z = 3, K = 16)$

CHAPTER

FIVE

STATE SPACE COMPUTATIONS

Now we describe the implementation for the key tasks involved so far. We will need to:

- compute state state partitions: $D := \cup_{k=1}^{K} Q_k$;
- construct transitions from each subspace Q_k into corresponding sets $P(a)(Q_k) \subset D$, for every possible action profile $a \in A$;
- record intersections $P(a)(Q_k) \cap Q'_k$, for every $k, k' \in \mathbf{K}$; and
- sample uniform points from each Q_k, and check for nonempty samples that end up transitioning to each respective intersecting set $P(a)(Q_k) \cap Q'_k$.

5.1 Storage

We only need to store:

1. each index $k' \in \mathbf{I}(a, k) \subseteq \mathbf{K}$, which refers to some partition element(s) $Q_{k'}$ whose subset $Poly_{k'}$ is accessible from Q_k given operator $P(a)$.
   - This suffices for indexing the correct slices of equilibrium payoff sets over the corresponding subset $Poly_{k'}$ of the state space D.
2. the finite number of vertices of each $Poly_{k'}$ and the corresponding linear (weak) inequality representation of each $Poly_{k'}$.
   - This will become apparent later when we solve *separable* bilinear programming problems where it involves optimizing over these subsets $Poly_{k'}$ (when constructing max-min punishment values in the game).
3. the sub-samples from all the Hit-and-Run realizations that belong to every partition element Q_k which will end up in particular intersections summarized by each polytope $Poly_{k'}$.

5.2 Implementation

Since we have finite partition elements Q_k and finite action profile set $A \ni a$, then we can enumerate and store all intersections previously denoted by $\{Poly_{k'(k,a)} : \forall a \in A, k \in \mathbf{K}\}$ or equivalently by their index sets $\{\mathbf{I}(a, k) : (a, k) \in A \times \mathbf{K}\}$.

```
Pseudocode

For each $(a,k) \in A \times \mathbf{K}$:

  • Get vertex representation of $Q_k \in D$
  • Set $P(a)$ as $P$
  • Get vertex representation $T$ from $P(Q_k)$
  • Simulate Hit-and-Run uniform realizations in simplex $T$. Get

        $X := \{\lambda_n\}_{n=1}^{N_{sim}} \leftarrow RandomPolyFill(N_{sim}, T)$

  • Set $i \leftarrow 0$
      For each $k' \in \mathbf{K}$:
      – Get vertex representation of $Q'_k \in D$
      – Get intersection $InterPoly = Q_{k'} \cap T$ (a polytope) as:

            $InterPoly \leftarrow PolyBool(Q_{k'}, T)$

      – If $InterPoly \neq \emptyset$:
          * Set $i \leftarrow i + 1$
          * Store index to partition elements $Q_{k'}$ when $InterPoly$ is nonempty. Set

                $TriIndex(a,k)(i) \leftarrow k'$

          * Store vertex data of polytope. Set:

                $PolyVerts(a,k,k') \leftarrow Interpoly$

          * Store linear inequality representation of the same polytope. Set:

                $PolyLcons(a,k,k') \leftarrow Vert2Lcon(InterPoly)$

          * Map Monte Carlo realizations $X$ under operator $P$. Set:

                $Y \leftarrow P(X)$

          * List members of $Y$ that end up in $InterPoly$. Set:

                $Y_{in} \leftarrow InPolytope(Y, InterPoly)$
                $\{PolyRand(a,k,k'), in\} \leftarrow Y_{in}$

          * Record all vectors $\{\lambda_n\} \subseteq X$ that induce $Y_{in}$ under map $P(a)$:

                $\{PolyQjRand(a,k,k'), Index\} \leftarrow X(in)$

          * Store corresponding indices $\{n : \lambda_n P(a) \in Y_{in}\}$:

                $PolyIndexQjRand(a,k,k') \leftarrow Index$

  • Return: $TriIndex, PolyVerts, PolyLcons, PolyRand, PolyQjRand, PolyIndexQjRand$
```

Note: Computationally, we only need to construct sets $\{Poly_{k'(k,a)} : \forall a \in A, k \in \mathbf{K}\}$ (e.g. *PolyVerts* and *PolyLcons* in the pseudocode above) or $\{\mathbf{I}(a,k) : (a,k) \in A \times \mathbf{K}\}$ (i.e. *TriIndex* in the pseudocode above) once beforehand.

Relevant functions ▶

`Simplex_IntersectPmap` (*self*)

Returns 4 possible output:

- *QP* :
  - a structure variable containing all others below.
- *TriIndex* :
  - a cell array containing indices $k'(a,k) \in \mathbf{I}(a,k) \subseteq \mathbf{K}$ of partition elements that have non-empty intersections with each simplicial image $P(a)(Q_k)$.
- *PolyVerts* :
  - a cell array, where each cell is an array with rows corresponding to vertices of $Poly_{k'(a,k)}$, a polytope contained in the partition element Q_k. Each cell element is consistent with the index $k'(a,j) \in \mathbf{I}(a,k) \subseteq \mathbf{K}$ stored in *TriIndex*.
- *PolyLcons* :
  - is a set of linear (weak) inequality constraint representation of *PolyVerts*.
- *PolyRand* :
  - Realizations of random vectors $\lambda_s P(a) \in Q_{k'}$ where $k' \in \mathbf{I}(a,k)$ and $\lambda_s \sim \mathbf{U}[Q_k]$— i.e. is uniformly drawn from Q_k according to a Hit-and-Run algorithm [HYPERLINK TO ALGORITHM DESCRIPTION], classified according to each *PolyVerts{a}{k}{k'}*.
- *PolyQjRand* :
  - Inverse of *PolyRand*. Each *PolyQjRand{a}{k}{k'}* gives the set of $\lambda_s \sim \mathbf{U}[Q_k]$, where under action profile a, the induced vector is $P(a)(\lambda_s) \in Q_{k'}$ and $k' \in \mathbf{I}(a,k)$.
- *PolyIndexQjRand* :
  - Each *PolyIndexQjRand{a}{j}* gives the set of indices $\{s \in \{1,...,N_{sim}\} : s \mapsto \lambda_s \in Q_k, \lambda_s P(a) \in Q_{k'}$ and $k' \in \mathbf{I}(a,k)\}$. The number of *Monte Carlo simulations* of these uniform vectors subject to each convex body Q_k has to be prespecified as N_{sim}.

See also:
PolyBool, **Simplex_Intersect**, **Vert2Lcon**, **RandPolyFill**, **cprnd**

CHAPTER

SIX

EQUILIBRIUM PAYOFF CORRESPONDENCE

In *State Space* we constructed a partition of the simplex $D := \Delta(\mathcal{Z})$. Now, we let D be the domain of the equilibrium payoff correspondence. The task ahead is to approximate the equilibrium value correspondence $\mathcal{V} : D \rightrightarrows \mathbb{R}^{\mathcal{Z}}$ using convex-valued step correspondences.

6.1 Background

Notation reminder:

- Action profile of small players on $[0, 1]$, $a \in A$. (Assume A is a finite set.) Each small player takes on a *personal state* $j \in \mathcal{Z} := \{-N, ..., -1, +1, ..., +M\}$ at each date $t \in \mathbb{N}$.
- Actions of large player (G), $b \in B$. $B := \{b \in \mathbb{R}^{\mathcal{Z}} : -m \leq b(j) \leq \bar{m},$ for $j > 0,$ and, $0 \leq b(j) \leq \bar{m},$ for $j < 0, \forall j \in \mathcal{Z}\}$ is a finite set and contains vectors b that are physically feasible (but not necessarily government-budget feasible in all states).
- Extended payoff vector space, $\mathbb{R}^{\overline{\mathcal{Z}}}$, where $\overline{\mathcal{Z}} := \mathcal{Z} \cup \{G\}$.
- Probability distribution of small players on finite set $\mathcal{Z}$, $\lambda \in D := \Delta(\mathcal{Z})$.
- Profile of continuation values of agents, $w \in \mathbb{R}^{\mathcal{Z}}$.
- Transition probability matrix at action profile a, $P(a)$
- Individual $j \in \mathcal{Z}$, given action $a(j)$ faces transition probability distribution, $p^j(a(j)) \in P(a)$
- Flow payoff profile, $v_j(a, b) := u(c^b(j)) - \phi(a_j)$, where
  - $v(a, b) := (v_j(a, b))_{j \in \mathcal{Z}}$;
  - Utility-of-consumption function, $u(\cdot)$; and
  - Disutility-of-effort/action function, $-\phi(\cdot)$.
- Public date-t history, $h^t := \{\lambda^t, x^t, b^{t-1}\}_{t \geq 0}$, where
  - $\lambda^t = (\lambda_0, ..., \lambda_t)$ is a history of agent distributions up to and include that of date t;
  - $x^t = (x_0, ..., x_t)$ where x_t is a date-t realization of the random variable $X_t \sim_{i.i.d.} \mathbf{U}([0, 1])$; and
  - $b^{t-1} = (b_0, ..., b_{t-1})$ is a history of government policy actions up to the end of date $t - 1$ and let $\{b_{-1}\} = \emptyset$.

  Also, $h^0 := (\lambda_0, x_0)$.

Definition (Consistency)

Let $\mathcal{W} : D \rightrightarrows \mathbb{R}^{\overline{\mathcal{Z}}}$ be a compact- and convex-valued correspondence having the property that $w(G) = \sum_{j\in\mathcal{Z}} \lambda(j)w(j)$ for all $(\lambda, w) \in$ graph$(\mathcal{W})$. A vector $(b, a, \lambda', w) \in B \times A \times \Delta(\mathcal{Z}) \times \mathbb{R}^{\overline{\mathcal{Z}}}$ is **consistent with respect to** $\mathcal{W}$ at λ if

1. $-\lambda b^T \geq 0$;
2. $\lambda' = \lambda P(a)$;
3. $w \in \mathcal{W}(\lambda')$; and
4. For all $j \in \mathcal{Z}$, $a(j) \in \text{argmax}_{a'} \left\{(1-\delta)\left[u(c^b(j)) - \phi(a')\right] + \delta \mathbb{E}_{p^j(a')}[w(i)]\right\}$.

Definition (Admissibility)

For $(\lambda, b) \in D \times B$ **let**

$$\pi(\lambda, b) := \min_{(a',\lambda'',w')} \left[(1-\delta)\sum_{j\in\mathcal{Z}} \lambda(j)[u(c^b(j)) - \phi(a'(j))] + \delta \sum_{j\in\mathcal{Z}} \lambda''(j)w'(j)\right],$$

subject to (b, a', λ'', w') is consistent with respect to $\mathcal{W}(\lambda)$. Let $(\tilde{a}(\lambda, b), \tilde{\lambda}'(\lambda, b), \tilde{w}(\lambda, b))$ denote the solutions to the corresponding minimization problem. A vector $(b, a, \lambda', w) \in B \times A \times D \times \mathbb{R}^{\overline{\mathcal{Z}}}$ is said to be **admissible with respect to** $\mathcal{W}(\lambda)$ if

1. (b, a, λ', w) is consistent with respect to $\mathcal{W}(\lambda)$; and
2. $(1-\delta)\sum_{j\in\mathcal{Z}} \lambda(j)[u(c^b(j)) - \phi(a(j))]\delta \sum_{j\in\mathcal{Z}} \lambda'(j)w(j) \geq \max_{b'\in B(\lambda)} \pi(\lambda, b')$, where $B(\lambda) := \{b \in B : -\lambda b^T \geq 0\}$.

Admissible payoff vectors

The payoff vector defined by an admissible vector (b, a, λ', w) at λ is given by

$$E_G(b, a, \lambda', w)(\lambda) = (1-\delta)\sum_{j\in\mathcal{Z}} \lambda(j)[u(c^b(j)) - \phi(a(j))] + \delta \sum_{j\in\mathcal{Z}} \lambda'(j)w(j)$$
$$E_j(b, a, \lambda', w)(\lambda) = (1-\delta)\left[u(c^b(j)) - \phi(a(j))\right] + \delta \mathbb{E}_{p^j(a(j))}[w(i)].$$

Note that $E_G(b, a, \lambda', w)(\lambda) = \sum_{j\in\mathcal{Z}} \lambda(j)E_j(b, a, \lambda', w)(\lambda)$.

In the paper, we proved the following:

SSE Recursive Operator

A SSE is a strategy profile $\sigma \equiv \{\alpha_t, \beta_t\}_{t\geq 0}$ such that given initial game state λ_0, for all dates $t \geq 0$, and all public histories $h^t := \{\lambda^t, x^t, b^{t-1}\}_{t\geq 0}$, $a := \alpha_t(h^t, b_t)$ and $b := \beta_t(h^t)$, and, if $\mathcal{V}$ is the SSE payoff correspondence, then $\mathcal{V}$ is the largest fixed point that satisfies the recursive operator

$$\mathbf{B}(\mathcal{V})(\lambda) := \text{co}\{E(b, a, \lambda', w)(\lambda) \,|\, (b, a, \lambda', w) \text{ is admissible w.r.t.} \mathcal{V}(\lambda)\},$$

where co denotes the convex hull of a set.
The object of interest can be found recursively: $\mathcal{V} = \lim_{n\to+\infty} \mathbf{B}^n(\mathcal{W}_0)$, for any initial convex-valued and compact correspondence $\mathcal{W}_0$.

Note: Given state λ and agent payoff vector $w \in \mathbb{R}^{\mathcal{Z}}$ determine a unique corresponding government payoff given by $\lambda \cdot w$. We can thus ignore the government payoff when defining the equilibrium value correspondences and their approximations, and restrict their codomain to $\mathbb{R}^{\mathcal{Z}}$.

CHAPTER SEVEN

APPROXIMATING SSE OPERATORS

The goal ahead is to *approximate* and provide computable representations of:

- each candidate correspondence $\mathcal{W} : D \rightrightarrows \mathbb{R}^{\mathcal{Z}}$; and
- the operator $\mathcal{W} \mapsto \mathbf{B}(\mathcal{W})$.

7.1 Conceptual

Recall $\{Q_k \,|\, k = 1, \ldots, K\}$ denotes a partition of D, so $D = \bigcup_{k=1}^{K} Q_k$. An upper hemicontinuous, compact- and convex-valued correspondence $\mathcal{W} : D \rightrightarrows \mathbb{R}^{\mathcal{Z}}$ can be approximated by step-valued correspondences using the following procedures: Letting

$$\omega_k^o(\lambda) := \begin{cases} \text{co} \bigcup_{\lambda \in Q_k} \mathcal{W}(\lambda') & \text{if } (\lambda) \in Q_k, \\ \emptyset & \text{otherwise,} \end{cases}$$

the correspondence defined by $\mathcal{W}^o(\lambda) := \bigcup_k \omega_k^o(\lambda)$ gives an *outer step-valued approximation* of $\mathcal{W}$.

Similarly, letting

$$\omega_k^i(\lambda) := \begin{cases} \bigcap_{\lambda \in Q_k} \mathcal{W}(\lambda) & \text{if } \lambda \in Q_k, \\ \mathbb{R}^{\mathcal{Z}} & \text{otherwise,} \end{cases}$$

the correspondence defined by $\mathcal{W}^i(\lambda) := \bigcap_k \omega_k^i(\lambda)$ yields an *inner step-valued approximation* of $\mathcal{W}$.

7.2 Practical

Since the convex-valued approximations $\mathcal{W}^o$ and $\mathcal{W}^i$ are constant on each partition element Q_k, and there are $K < +\infty$ partition elements, these approximations can be further approximated by constructing outer and inner approximations for the sets $\omega_k^o(\lambda)$ and $\omega_k^i(\lambda)$ using **convex polytopes**. Let $\mathbb{S}^{N_z - 1} := \left\{x \in \mathbb{R}^{N_z} : \|x\| = 1\right\}$ be the unit $(N_z - 1)$-sphere where the norm $\|\cdot\|$ is given by $\|x\|_2 = \left(\sum_{j=1}^{N_z} x_j^2\right)^{1/2}$. Suppose we have finite sets of directional vectors: $H := \{h_l \in \mathbb{S}^{N_z-1} : l = 1, ..., L\}$ and $\tilde{H} := \{\tilde{h}_l \in \mathbb{S}^{N_z-1} : l = 1, ..., L'\}$. Let $\bar{\omega}_k^o(\lambda)$ and $\bar{\omega}_k^i(\lambda)$ denote the corresponding polytope approximations, respectively, of $\omega_k^o(\lambda)$ and $\omega_k^i(\lambda)$, where

$$\bar{\omega}_k^o(\lambda) := \begin{cases} \bigcap_{l=1}^{L} \{z | h_l \cdot z \leq c_l^o(k)\} & \text{if } \lambda \in Q_k, \\ \emptyset & \text{otherwise} \end{cases},$$

and,

$$\bar{\omega}_k^i(\lambda) := \begin{cases} \bigcap_{l=1}^{L'} \{z | \tilde{h}_l \cdot z \leq c_l^i(k)\} & \text{if } \lambda \in Q_k, \\ \emptyset & \text{otherwise} \end{cases}.$$

Let $\bar{\mathcal{W}}^o := \cup_{k \in \mathbf{K}} \bar{\omega}_k^o$ and $\bar{\mathcal{W}}^i := \cup_{k \in \mathbf{K}} \bar{\omega}_k^i$ denote the resulting correspondences. One would like the "true" correspondence $\mathcal{W}$ to be "sandwiched" by polytope "step-correspondences" $\bar{\mathcal{W}}^o$ from the outside, and, by $\bar{\mathcal{W}}^i$

from the inside. [1]

$$\bar{\mathcal{W}}^i \subset \mathcal{W}^i \subset \mathcal{W} \subset \mathcal{W}^o \subset \bar{\mathcal{W}}^o. \tag{7.1}$$

The last statement (7.1) is only true if the step-correspondence levels $c_l^o(k)$ and $c_l^i(k)$ are defined, respectively, as the maximal and minimal levels over each domain partition element Q_k, in each direction $h_l \in H$ or $\tilde{h}_l \in \tilde{H}$. [2]

In the next section, we show how to construct these upper- and lower bounding estimates $c_l^o(k)$ and $c_l^i(k)$ by using stochastic global optimization programs and also *separable* bilinear program formulations, when $\mathcal{W}$ represents a candidate guess of the *symmetric sequential equilibrium* payoff correspondence in our class of games.

[1] This idea of providing both upper- and lower-bounding estimates of a given correspondence was first proposed by [JYC2003] in the computation of repeated games. Our proposed method is a modification of [SY2000] who in turn extended [JYC2003] to the computation of value correspondences in dynamic games. Our contribution will be in the form of *bilinear programming formulations* as a practical and computable means of constructing these approximate correspondences.

[2] In the context of our game, where $\mathcal{W}$ stands for a candidate guess of the equilibrium value correspondence, the last statement (7.1) is only true if the step-correspondence levels $c_l^o(k)$ and $c_l^i(k)$ are defined, respectively, as the **globally** maximal and minimal values of each nonlinear programming problem (which is defined over each state-space partition element Q_k, in each direction $h_l \in H$ or $\tilde{h}_l \in \tilde{H}$) that summarizes the concept of *symmetric sequential equilibrium* of the game.

CHAPTER

EIGHT

BILINEAR PROGRAMS AND SSE OPERATOR

Here we give an overview of our main computational insight and proposed method for constructing the operators $\bar{\mathcal{W}}^o \mapsto \mathbf{B}^o(\bar{\mathcal{W}}^o)$, and $\bar{\mathcal{W}}^i \mapsto \mathbf{B}^i(\bar{\mathcal{W}}^i)$.

Given a candidate correspondence $\mathcal{W}$, evaluating the *symmetric sequential equilibrium* (SSE) operator at this point in the set of compact and convex-valued correspondences (see Definition 3 in the paper), $\mathbf{B}(\mathcal{W})$, will involve:

1. Calculating state-dependent max-min punishment values, $\pi(\lambda) := \max_b \pi(\lambda, b)$.
   - We show that this is amenable to a *separable* bilinear program (BLP).
   - We will describe how these BLPs are solved to ϵ-global optimality.
2. Given $\pi(\lambda)$, compute the total-payoff sets supported by action-states-continuation-value tuples, (b, a, λ, w), that are *admissible* with respect to $\mathcal{W}$:
   - We will show that this consists of subproblems that are *non-separable* BLPs.
   - These can be solved by a specific stochastic global optimization problem that involves sub-problems that are *linear programs* (LP).

We adapt Steps 1 and 2 above for both outer- and inner-approximations, respectively, yielding approximate outer- and inner evaluations of the step-correspondence images $\mathbf{B}^o(\bar{\mathcal{W}}^o)$ and $\mathbf{B}^i(\bar{\mathcal{W}}^i)$.

CHAPTER

NINE

PUNISHMENT VALUES AND BLPS

The first step is to construct punishment values over each partition element of the state space. This will turn out to be amenable to *separable bilinear programming* formulations.

Punishment payoffs for the large player G are constructed as follows.

- For each $a \in A$, define a correspondence $\tilde{\mathcal{W}} : D \times A \rightrightarrows \mathbb{R}^{\mathcal{Z}}$ by

$$\begin{aligned}\tilde{\mathcal{W}}(\lambda', a) := \{w \in \mathcal{W}(\lambda') \,|\, \forall j \in \mathcal{Z}, \forall a' \in \tilde{A} \\ \delta[p^j(a') - p^j(a_j)] \cdot w \leq (1-\delta)[\phi(a') - \phi(a_j)]\}.\end{aligned}$$

Note: Note $\tilde{\mathcal{W}}$ encodes two requirements:

- Continuation values w must be *consistent* with the set $\tilde{\mathcal{W}}(\lambda')$ where w itself enforces the action-continuation-state profile pair $(a, \lambda') \equiv (a, \lambda P(a))$; and
- individually, the action-and-promised-value pair (a_j, w_j) are optimal (i.e. "incentive compatible").

- Next, construct government punishment vectors
  - $(\check{\pi}_k)_{k=1}^K$, and
  - $(\hat{\pi}_k)_{k=1}^K$

  by letting

$$\begin{aligned}\pi(\lambda) := \max_{b \in B} \min_{a, \lambda', w} & [(1-\delta)\lambda \cdot v(a,b) + \delta\lambda' \cdot w], \\ \text{s.t. } & \lambda' = \lambda P(a), \\ & -\lambda b^T \geq 0, \\ & w \in \tilde{\mathcal{W}}(\lambda', a),\end{aligned} \tag{9.1}$$

  Then define: $\check{\pi}_k := \min_{\lambda \in Q_k} \pi(\lambda)$ and $\hat{\pi}_k := \max_{\lambda \in Q_k} \pi(\lambda)$.

Note:

- If $\mathcal{W}(\lambda')$ is defined as a convex polytope, then $\tilde{\mathcal{W}}(\lambda', a)$ is also a convex polytope.
- Given $(a, b) \in A \times B$, the minimization program (9.1) is a mild nonlinear programming problem–i.e. a *separable bilinear programming formulation*–of the following generic form

$$\begin{aligned}\min_{\lambda, w} & u^T + \lambda^T Q w \\ s.t. & \lambda \in \mathcal{F}_1, \\ & w \in \mathcal{F}_2.\end{aligned} \tag{9.2}$$

- Moreover, $\mathcal{F}_1$ and $\mathcal{F}_2$ are disjoint convex and bounded polytopes.

Existence of Optimum

If F_1 and F_2 are bounded then there exists an optimal solution of (9.2), (λ^*, w^*), such that $\lambda^* \in \mathcal{F}_1$ and $w \in \mathcal{F}_2$.
See [HPT2000] or [HT1996].

Separable BLP is NP-complete

A bilinear program can be solved in NP time.
See [Man1995].

Assume that we are solving the minimization problem in (9.2). We employ a well-known deterministic global optimization algorithm known as branch-and-bound (BNB). First, a *relaxation* of the bilinear program is solved. Typically, this is done by solving inexpensive LPs. [CM2009] The solution of the relaxed problem yields a lower bound on the globally optimal solution which is a convex lower envelope. The relaxation is then iteratively refined, by refining the domain (feasible sets) and successively eliminating dominated local optima. (This is also a common method in solving integer linear programs.) An upper bound estimate of the optima can be found by using local nonlinear solvers (e.g. SNOPT and IPOPT) over each branch. Thus we have successively improved branching partitions of the domain (i.e. *branching*) and lower- and upper-bounding estimates (i.e. *bounding*) of the ϵ-global optimum.

Note: The BNB algorithm we use follows [McC1976] and is implemented through the Bilinear Matrix Inequality BNB interface (BMIBNB) available in Stefan Lofberg's YALMIP (http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Solvers.BMIBNB). To solve the local lower-bounding LPs, we use the GNU GLPK (http://www.gnu.org/software/glpk/) open-source optimizer and to solve the upper-bounding nonlinear programs, we use either SNOPT (http://www.sbsi-sol-optimize.com/asp/sol_products_snopt_desc.htm) or IPOPT (https://projects.coin-or.org/Ipopt).

9.1 Implementing punishment values

The following pseudocode implements the punishment calculations for the outer-approximation scheme:

Pseudocode

```
Input: $H, c, PolyLcons$

For each $(Q_k, a, b) \in D \times A \times B$:

    • Markov map $P \leftarrow P(a)$
    • Simplex $T \leftarrow P(Q_k)$
    • Get $\mathbf{I}(a,k) := \{k' \in \mathbf{K} : k' \mapsto Q_{k'} \in D, Q_{k'} \cap T \neq \emptyset \}$
        – Uses: xTriIndex from Simplex_IntersectPmap

        For each $k' \in \mathbf{I}(a,k)$:

            – Get linear inequality representations of $T \cap Q_{k'}$ as

                $(M_{k'}, d_{k'}) \leftarrow xPolyLcons(a, k, k')$

            – Get current payoff profile $v(a,b)$
            – Solve separable BLP

                $\breve{\pi}(Q_k, a, b, k') = \min_{\lambda \in Q_k, w} \lambda[(1-\delta)v(a,b) + \delta P w]$
                $s.t.$
                $M_{k'}(\lambda P)^T \leq d_{k'}$
                $\lambda b^T \leq 0$
                $Hw \leq c^o(Q'_k, \cdot)$
                $\delta[p^j(a') - p^j(a_j)] \cdot w \leq (1-\delta)[\phi(a') - \phi(a_j)], \forall j \in \mathcal{Z}$

    • Get $\breve{\pi}(Q_k, a, b) = \min_{k' \in \mathbf{I}(a,k)} \breve{\pi}(Q_k, a, b, k')$.
  • Get $\breve{\pi}(Q_k) = \max_{b \in B} \min_{a \in A} \breve{\pi}(Q_k, a, b)$.
```

Note:

- The *separable* constraint sets $\mathcal{F}_1$ and $\mathcal{F}_2$ for λ and w, respectively, are given by constraints in the BLP. These constraint say the following.
  - $\mathcal{F}_1$: the first two constraints require $\lambda \in Q_k$ to be such that
    - for each $a \in A, k \in \mathbf{K}$ and $k' \in \mathbf{I}(a,k)$, the resulting continuation state $\lambda P(a) \in Q_{k'} \cap P(Q_k)$; and
    - given a fixed policy b, the choice over λ renders b feasible according the the government budget constraint;
  - $\mathcal{F}_2$ is given by the requirements that w be
    - consistent with respect to the step correspondence slice $\mathcal{W}(k')$ which has constant levels over the partition element $Q_{k'}$; and
    - such that w is incentive compatible for all small agents.
- Constructing the punishment values $\hat{\pi}_k$ for the *inner-approximation* scheme is similar to what we did above in detail for $\breve{\pi}_k$. The only differences are

1. in the second last step of the pseudocode above, replace that line with:

$$\hat{\pi}(Q_k, a, b) = \max_{k' \in \mathbf{I}(a,k)} \hat{\pi}(Q_k, a, b, k').$$

Of course we should also re-label all the $\check{\pi}$ notation for the punishment value function with $\hat{\pi}$; and

2. *maximize* over $\lambda \in Q_k$ in the main BLP problem.

Relevant functions ▶

`Punish_Outer` (*self*)

Returns:

- *pival* :
  - A $(K \times 1)$ numeric array containing elements $\check{\pi}(Q_k)$ where $k \in \mathbf{K}$.

See also:

PunishK

`Punish_Inner` (*self*)

Returns:

- *pival* :
  - A $(K \times 1)$ numeric array containing elements $\hat{\pi}(Q_k)$ where $k \in \mathbf{K}$.

See also:

PunishK

CHAPTER

TEN

COMPUTING APPROXIMATE SSE PAYOFFS

Now we are ready to describe the computation of the approximate SSE payoff correspondence. The basic idea is from [JYC2003] who use *linear program* (LP) formulations as the approximation. Our extension illustrates that when we have probability distributions (with finite support) as state variables, the approximate SSE payoff correspondence can be constructed via *bilinear program* (BLP) formulations.

Notation:

- Let

$$v_j(a,b) := u(c^b(j)) - \phi(a_j)$$

- Given:
  - a vector of agent actions a,
  - a government policy vector b, and
  - a vector of continuation payoffs w,

  the vector of agents' expected payoffs is defined by

$$E(a,b,w) := ((1-\delta)v_j(a,b) + \delta P^j(a_j) \cdot w)_{j \in \mathcal{Z}}.$$

10.1 Outer Approximation: Conceptual

We can now define the outer approximation $\mathbf{B}^o(\mathcal{W})$.

- For each search subgradient $h_l \in H$ and each partition element Q_k, let

$$\begin{aligned} c_l^{o+}(k) := \max_{(a,b)\in A\times B, \lambda \in Q_k, w} & [h_l \cdot E(a,b,w)], \\ \text{s.t. } & \lambda' = \lambda P(a), \\ & \lambda b^T \leq 0, \\ & w \in \tilde{\mathcal{W}}(\lambda', a), \\ & (1-\delta)\lambda \cdot v(a,b) + \delta \lambda' \cdot w \geq \check{\pi}_k, \end{aligned} \tag{10.1}$$

- **Then define**

$$\bar{\omega}_k^{o+}(\lambda) := \begin{cases} \bigcap_{l=1}^L \{z \mid h_l \cdot z \leq c_l^{o+}(k)\}, & \text{if } \lambda \in Q_k, \\ \emptyset, & \text{otherwise.} \end{cases}$$

Note: Since $A \times B$ is a finite set of action profiles, we can evaluate the program (10.1) as a special class of a nonlinear optimization problem–a *nonseparable* bilinear program (BLP)–for each fixed $(a,b) \in A \times B$. Then we can maximize over the set $A \times B$, by table look-up.

10.2 Outer Approximation: Implementation

Now we deal with implementing the idea in *Outer Approximation: Conceptual*. The outer-approximation scheme to construct $\mathbf{B}^o(\mathcal{W}^o)$ in the set of problems in (10.1) is computable by following the pseudocode below:

Pseudocode

```
Input: $H, c, \hat{\pi}, Poly$

For each $(Q_k, h_l, a) \in D \times H \times A$:

    • Markov map $P \leftarrow P(a)$
    • Simplex $T \leftarrow P(Q_k)$
    • Get Hit-and-Run uniform draws constrained to be in $Q_k$: $X := \{\lambda_n\}_{n=1}^{N_{sim}} \leftarrow RandomPolyFill(N_{sim}, T)$
    • Get feasible set $F \leftarrow F(B, Q_k) := \{(\lambda, b) \in Q_k \times B : \lambda \in X \subseteq Q_k, -\lambda b^T \geq \mathbf{0}\}$
    • Get $\mathbf{I}(a,k) := \{k' \in \mathbf{K} : k' \mapsto Q_{k'} \in D, Q_{k'} \cap T \neq \emptyset\}$
        – Uses: TriIndex from Simplex_IntersectPmap

        For each $k' \in \mathbf{I}(a,k)$:

        – Get relevant feasible policy set $F(k'; a, k) \leftarrow \{(\lambda, b) \in F : \lambda P \in Q_{k'}\}$
            For each $(\lambda, b) \in F(k'; a, k)$:
            ∗ Get current payoff profile $v(a, b)$
            ∗ Solve conditional LP:

              $c_+^o(Q_k, h_l, a, k', \lambda, b) = \max_w h_l[(1-\delta)v(a,b) + \delta P w]$
              $s.t.$
              $Hw \leq c^o(Q_{k'}, h_l)$
              $\delta[p^j(a') - p^j(a_j)] \cdot w \leq (1-\delta)[\phi(a') - \phi(a_j)], \forall j \in \mathcal{Z}$
              $-\lambda \delta P w \leq \lambda[(1-\delta)v(a,b)] - \check{\pi}(Q_k)$

        – Get $c_+^o(Q_k, h_l, a, k') = \max_{(\lambda,b)\in F(k';a,k)} c_+^o(Q_k, h_l, a, k', \lambda, b)$.
    • Get $c_+^o(Q_k, h_l, a) = \max_{k' \in \mathbf{I}(a,k)} c_+^o(Q_k, h_l, a, k')$.
• Get $c_+^o(Q_k, h_l) = \max_{a \in A} c_+^o(Q_k, h_l, a)$.
```

Note:

- In the pseudocode, we can see that for every fixed $(Q_k, a, k') \in D \times A \times \mathbf{I}(a,k)$ and every feasible b, the nested family of programming problems are *nonseparable bilinear programs* (BLP) in the variables (λ, w).
- The inner most loop thus implements our Monte Carlo approach to approximately solve for an ϵ-global solution to the *nonseparable* BLPs.
- Conditional on each draw of λ, this becomes a standard linear program (LP) in w within each innermost loop of the pseudocode.
- Given the set of subgradients H, an outer-approximation update on the initial step correspondence $\mathcal{W}^o$, is now sufficiently summarized by $\mathcal{W}_+^o = \mathbf{B}^o(\mathcal{W}^o) \leftarrow (H, c_+^o(Q_k, h_l), \mathcal{W}^o)$.

Relevant functions ▶

`Admit_Outer_LPset` (*self*)

Returns:

- *Cnew* :
  - A $(L \times K)$ numeric array containing elements $c_{+}^{o}(Q_k, h_l)$ where $k \in \mathbf{K}$ and $h_l \in H$ are, respectively, a partition element of the correspondence domain D, and, search subgradient in direction indexed by $l \in \{1, ..., |H|\}$.

See also:

Punish_Outer

10.3 Inner Approximation: Conceptual

We now define the *inner approximation* of the SSE value correspondence operator as $\mathbf{B}^i(\mathcal{W})$ below.

- Denote $\tilde{H}(k)$ as a finite set of $\tilde{L}(k)$ spherical codes (to be used as *approximation subgradients*, where each element is $\tilde{h}_l(k)$ and $\|\tilde{h}_l(k)\|_2 = 1$ for all $l = 1, ..., \tilde{L}$ and $k \in \mathbf{K}$.
- Assume an initial inner step-correspondence approximation of some convex-valued and compact-graph correspondence

$$\mathcal{W}^i := \bigcup_{Q_k \in D} \bigcap_{\tilde{h}_l(k) \in \tilde{H}(k)} \left\{ z \in \mathbb{R}^{\mathcal{Z}} : \tilde{h}_l(k) z \leq c(Q_k, \tilde{h}_l) \right\}.$$

- Define another finite set of fixed L *search subgradients*, made up also of spherical codes, H, just as in the outer approximation method above. [1]
- For each *search subgradient* $h_l \in H$ and each partition element Q_k, let

$$\begin{aligned} V_l^{i+}(k) := \min_{\lambda \in Q_k} \max_{(a,b) \in A \times B, w} & [h_l \cdot E(a, b, w)], \\ \text{s.t. } & \lambda' = \lambda P(a), \\ & \lambda b^T \leq 0, \\ & w \in \tilde{\mathcal{W}}^i(\lambda', a), \\ & (1 - \delta)\lambda \cdot v(a, b) + \delta \lambda' \cdot w \geq \hat{\pi}_k, \end{aligned} \tag{10.2}$$

  Set $V_l^{i+}(k) = -\infty$ if the optimizer set is empty.
- In contrast to *Outer Approximation: Conceptual*, obtain the following additional step.
  - Let $(a_l^*(k), b_l^*(k), w_l^*(k))$ denote the maximizers in direction h_l and over domain partition element Q_k, that induce the level $V_l^{i+}(k)$ above.
  - Then the corresponding vector of agent payoffs is

$$z_l^+(k) := E(a_l^*(k), b_l^*(k), w_l^*(k)).$$

  - Define the set of vertices $Z(k) = \{z_l^+(k) : l = 1, ..., L\}$ and let $\mathcal{W}^{i+}(k) = \text{co}(Z(k))$.

[1] Note that we have to let the *approximation subgradients* $\tilde{H}$ to possibly vary with domain partition elements Q_k, as opposed to fixed *search subgradients* in H used in the optimization step. This is because the former is endogenously determined by the extra convex hull operation taken to construct an inner step-correpondence $\mathcal{W}^i$ at each successive evaluation of the operator $\mathbf{B}^i$.

- Update

$$Z^{+}(k)=\left\{z_l^{+}(k)\in Z(k):z_l^{+}(k)\in\partial\mathcal{W}^{i+}(k)\right\};$$

and find approximation subgradients $\tilde{H}^{+}(k)=\{\tilde{h}_1^{+}(k),...,\tilde{h}_{\tilde{L}^{+}}^{+}(k)\}$ and constants $C^{+}(k)=\{c_1^{+}(k),...,c_{\tilde{L}^{+}}^{+}(k)\}$ such that

$$co(Z^{+}(k))=\bigcap_{l=1}^{\tilde{L}^{+}}\left\{z:\tilde{h}_l^{+}(k)z\leq c_l^{+}(k)\right\},$$

and $\mathcal{W}^{i+}=\cup_k co(Z^{+}(k))=\mathbf{B}^i(\mathcal{W}^i)$.

Note: As in the *outer approximation* methods, since $A\times B$ is a finite set of action profiles, we can evaluate the program (10.2) as a special class of a nonlinear optimization problem–a *nonseparable* bilinear program (BLP)–for each fixed $(a,b)\in A\times B$. Then we can maximize over the set $A\times B$, by table look-up. Thus, the only difference computationally in the *inner approximation* method is the extra step of summarizing each inner step-correspondence $\mathcal{W}^{i+}$ by updates on:

- approximation subgradients in each $\tilde{H}^{+}(k)$;
- levels in each $C^{+}(k)$; and
- vertices, $Z^{+}(k)$,

for every $k\in\mathbf{K}$.

Relevant functions ▶

`Admit_Inner_LPset` (*self*)

Returns:

- *Znew* :
  - A $(L\times K)$ numeric array containing elements $z_l^{+}(k)$ where $k\in\mathbf{K}$ and $l\mapsto h_l\in H$ are, respectively, a partition element of the correspondence domain D, and, approximation subgradient in direction indexed by $l\in\{1,...,L\}$.

See also:

Punish_Inner

CHAPTER

ELEVEN

SOFTWARE

HARDPIG relies on the following software:

- MATLAB (http://www.mathworks.com) platform
- YALMIP (http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Solvers.BMIBNB) BMIBNB global optimization solver
  - Bilinear Matrix Inequality Branch-and-Bound solver
- GNU GLPK (http://www.gnu.org/software/glpk/) linear programming solver (in ANSI C)
- SNOPT (http://www.sbsi-sol-optimize.com/asp/sol_products_snopt_desc.htm) general-purpose local optimization solver (Fortran)
  - MATLAB Executable binaries files (http://www.scicomp.ucsd.edu/ peg/Software.html) for up to 300 variables and 300 constraints available freely from Phillip E. Gill.

Source codes (in MATLAB) and executables (in C/Fortran) are available from the authors via email.

CHAPTER TWELVE

INDICES AND TABLES

- *genindex*
- *search*

[BM1993] Bennett, Kristin and Olvi L. Mangasarian (1993): “Bilinear Separation of Two Sets in n-Space”. *Computational Optimization and Applications*, 2.

[HT1996] Horst, Reiner and Hoang Tuy (1996): “Global Optimization: Deterministic Approaches”. Springer Verlag.

[KTB2011] Kroese, Dirk P., Thomas Taimre and Zdravko I. Botev (2011): *Handbook of Monte Carlo Methods*. Wiley Series in Probability and Statistics. Wiley.

[Lov1999] Lovasz, Laszlo (1999): “Hit-and-run Mixes Fast”. *Mathematical Programming*, Ser. A 86, 443-461.

[LV2003] Lovasz, Laszlo and Santosh Vempala (2003): “Hit-and-Run is Fast and Fun”, Technical Report, Microsoft Research, MSR-TR-2003-05.

[McC1976] McCormick, Garth P. (1976): “Computability of Global Solutions to Factorable Nonconvex Programs: Part I. Convex underestimating problems”. *Mathematical Programming*, 10, 147–175.

[Smi1984] Smith, Robert L. (1986): “Efficient Monte-Carlo Procedures for Generating Points Uniformly Distributed over Bounded Regions”. *Operations Research*, 32, 1296-1308.

[CM2009] Carpara, Alberto and Michele Monaci. “Bidimensional packing by bilinear programming”. *Mathematical Programming Series A*, 118, 75–108.

[HPT2000] Horst R, P. Pardalos and N. Thoai (2000): *Introduction to global optimization*. 2nd Edition, Boston: Springer, 2000.

[HT1996] Horst R, Hoang Tuy (1996): *Global Optimization*. 3rd Edition, New York: Springer.

[Man1995] Mangasarian, Olvi L. (1995): “The linear complementarity problem as a separable bilinear program”. *Journal of Global Optimization*, 12, 1–7.

[JYC2003] Judd, Kenneth L., Sevin Yeltekin and James Conklin (2003): “Computing Supergame Equilibria”. *Econometrica*, 71(4), 1239-1254.

[SY2000] Sleet, Christopher and Sevin Yeltekin (2000): “On the Computation of Value Correspondences”. Unpublished. KGMS-MEDS, Northwestern University.