MatTuGames: A Matlab Game Theory Toolbox

Version 1.9.2 (R2024b) 14-Feb-2025

Author: Holger I. Meinhardt

E-Mail: holger.meinhardt@partner.kit.edu

Institution: Karlsruhe Institute of Technology (KIT)

Some Auxiliary Files

FrameToImage	- Converts a frame to an image.
FrameToImage2	- Converts a frame to an image using MYAA.
PlayCoreMovie	- Plays a movie from a collection of frames.
SaveFrames	- Saves the frames of a movie to different file formats.
ToSimplex	- Projects data from 3d to 2d.
ginv	- Computes a general inverse.
myaa	- MYAA Render figure with anti-aliasing.
qrginv	- Computes a pseudo-inverse using a QR-method.
toSymbols	- Converts a vector (vec, n) into digits.
vtk_export	- Exportes the graphical raw data to VTK legacy format.

Scirpt File

corevert	- External bash script to call the cdd library.

Document Files

MatTuGames_Version_1.9.2.m	- Additions and changes in version 1.9.2
ReadMe.pdf	- Installation instruction (PDF)
ReadMe.md	- Installation instruction (Markdown Format)
getting_started.m	- Checks the installation
getting_started.out	- Reference results of getting_started
getting_started.md	- Reference results of getting_started (Markdown Format)
manual_mat_tugames.pdf	- Manual (PDF)
MatTuGames_References.md	- Bibliography (Markdown Format)
MatTuGames_References.pdf	- Bibliography (PDF)
testcase_graphics	- Checking basic graphic installation.

Graphic Example Files

core_exp.pdf	- Core plot example 3d.
core_exp_all.pdf	- Core plot example 3d.
core_exp_prk.pdf	- Core plot example 3d.
core_exp_prn.pdf	- Core plot example 3d.
core_exp_shap.pdf	- Core plot example 3d.

core_exp_sol_all.pdf - Core plot example 3d.

core_exp_sol_none.pdf - Core plot example 3d.

core_exp_sol_prk.pdf - Core plot example 3d.

core_exp_sol_prn.pdf - Core plot example 3d.

core_exp_sol_shap.pdf - Core plot example 3d.

manual_exp2_core01.p - Core plot example 3d.

Mathematica Symbolic Toolbox Functions to call the Mathematica Package TuGames

tug_AdjustedWorthVectors - Computes the adjusted worth vectors of k-convex games.

tug_AllAntiSurpluses - Computes the minimum surpluses.

tug_AllMaxSurpluses - Computes the maximum surpluses.

tug_AntiPreKernel - Computes an anti-pre-kernel point.

tug_AntiPreKernelQ - Checks if an imputation is an anti-pre-kernel point.

tug_AvConvexQ - Checks on average convexity.

tug_AverageConvexQ - Checks on average convexity.

tug_BalancedCollectionQ - Verifies if the induced collections are balanced.

tug_BalancedKSelectionQ - Checks if an imputation induces a k-balanced selection.

tug_BalancedSelectionQ - Checks if an imputation induces a balanced selection.

tug_Bankruptcy - Creates a modest bankruptcy game.

tug_BelongToCoreQ - Checks if an imputation belongs to the core.

tug_BestCoalToMatrix - Computes an equivalence matrix.

tug_Bsc - Returns the set of most effective coalitions.

tug_CharacteristicValues - Computes the characteristic values.

tug_Coal2Dec - List of proper coalitions in Mathematica order.

tug_CollectionBalancedQ - Checks if a collection is balanced.

tug_CollectionOfDecreasingExcess - Creates the collection of decreasing excesses.

tug_Concession - Computes the concession vector.

tug_ContestedGarment - Computes the contested garment.

tug_ConvexQ - Checks convexity.

tug_ConvexUnanConditionQ - Checks convexity while relying on the unanimity coordinates.

tug_CoreElementsQ - Checks if an imputation belongs to the core.

tug_CoreQ - Checks if the core is non-empty.

tug_CostSavings - Creates the cost savings game.

tug_CriticalVal - Computes some critical epsilon values.

tug_DetQuasiAvConvex - Determines a quasi average convex game.

tug_DetRandCoord - Returns random unanimity coordinates.

tug_DetUCoord - Determines the missing unanimity coordinates of size greater than 2.

tug_Disagreement - Computes the disagreement vector.

tug_DualGame - Creates the dual of a Tu-game.

tug_EpsCore - Computes the least core.

tug_EqClass - Determines the equivalence classes from the set of most effective coalitions.

tug_EvalSumMinCoord - Calculates at most (n-1) inequalities of the unanimity coordinates constraints of nonnegative sums.

tug_ExcessValues - Determines the excesses.

tug_FindPreKernel	- Computes a pre-kernel element.
tug_GameMonotoneQ	- Checks on monotonicity.
tug_Gap	- Computes the gap function.
tug_GrandCoalitionLargestValueQ	- Checks if the grand coalition has largest value.
tug_GreedyBankruptcy	- Creates the greedy bankruptcy game.
tug_HarsanyiDividends	- Creates the unanimity coordinates.
tug_ImpToVec	- Converts an imputation to a set of vectors.
tug_ImputationQ	- Checks if a payoff vector is an imputation.
tug_IntersectionOfMaxExcessSets	- Determines if the set of proper coalitions having largest excesses has an empty intersection.
tug_IntersectionUpperLowerSetQ	- Checks if the intersection of the lower and upper set is non-empty.
tug_Kernel	- Computes a kernel point.
tug_KernelCalculation	- Computes a or some kernel element(s).
tug_KernelImputationQ	- Checks if an imputation is a kernel point.
tug_KernelVertices	- Computes a kernel segment.
tug_LargestAmount	- Computes the largest amount.
tug_LeastCore	- Determine the least core.
tug_LexiCenter	- Computes the lexi center.
tug_LowerSetIncImputationQ	- Checks if the lower set is included in the imputation set.
tug_LowerSetQ	- Checks if an imputation belongs to the lower set.
tug_MKernel	- Determines a kernel point.
tug_MLExtension	- Computes the multi-linear extension.
tug_MargValue	- Determines the marginal contribution vector.

tug_MaxExcessBalanced - Checks if the maximum surpluses are balanced.

tug_MaxExcessSets - Computes the set of proper coalitions having largest excesses.

tug_MinExcessBalanced - Determines if the minimum surpluses are balanced.

tug_MinUnanimityCoordinates - Returns the minimum unanimity coordinates.

tug_Mnuc - Determines the nucleolus.

tug_MonotoneQ - Checks on monotonicity.

tug_Nuc - Computes the nucleolus.

tug_OneNormalization - Creates a one normalized game.

tug_PreKernel - Computes a pre-kernel element.

tug_PreKernelEl - Computes a pre-kernel element.

tug_PreKernelEqualsKernelQ - Checks if the pre-kernel coincides with the kernel.

tug_PreKernelQ - Checks if an imputation is a pre-kernel element.

tug_PreNuc - Computes the pre-nucleolus.

tug_ProperAmount - Computes the proper amount.

tug_Quota - Computes the quotas.

tug_ReasonableOutcome - Computes the reasonable outcome.

tug_ReasonableSet - Computes the reasonable set.

tug_ScrbSolution - Determines the Scrb solution.

tug_SetsToVec - Converts the set of most effective coalitions to a set of vectors.

tug_ShapleyValue - Determines the Shapley value.

tug_ShapleyValueML - Determines the Shapley value using multi-linear extension.

tug_SmallestContribution - Determines the smallest contribution vector.

tug_StrictlyConvexUnanConditionQ	- Examines the sufficient condition of convexity in terms of unanimity coordinates.
tug_SuperAdditiveQ	- Checks on super-additivity.
tug_SymGameSizeK	- Returns a special type of symmetric game.
tug_SymGameType2	- Returns a special type of symmetric game.
tug_SymGameType3	- Returns a special type of symmetric game.
tug_SymGameType4	- Returns a special type of symmetric game.
tug_TalmudicRule	- Computes the Talmudic distribution rule.
tug_TauValue	- Determines the Tau value.
tug_UnanAvConvexQ	- Checks if the coordinates satisfy the sufficient and necessary condition of average convexity.
tug_UnanConvexQ	- Checks if the coordinates satisfy the sufficient and necessary condition of convexity.
tug_UnanimityCoordinates	- Determines all unanimity coordinates of the game
tug_UpperSetIncImputationQ	- Checks if the upper set is included in the imputation set.
tug_UpperSetQ	- Checks if an imputation belongs to the upper set.
tug_UtopiaVector	- Computes the utopia payoff.
tug_ValueExcess	- Computes an objective function to compute a pre-kernel element.
tug_VerticesCore	- Determines the vertices of the core.
tug_WeaklySuperAdditiveQ	- Checks if the Tu-game is weakly super-additive.
tug_WeightedMajority	- Creates the weighted majority game.
tug_ZeroMonotoneQ	- Checks on zero-monotonicity.
tug_ZeroNormalization	- Creates the zero normalized game.
tug_ZeroOneNormalization	- Creates the zero-one normalized game.
tug_kCover	- Determines from the Tu-game the corresponding k-game.

Main Functions: Serial Computing

ADvalue - Computes the Aumann-Dreze value.

ANucAirportProb - Computes the anti-nucleolus from an airport capital cost problem.

AP_DummyPlayer_propertyQ - Checks if the solution x satisfies the AP-Dummy player property.

AP_DummyPlayers - Returns the player who are AP-Dummy players.

AP_NullPlayer_propertyQ - Checks if the solution x satisfies the AP-Null player property.

AP_NullPlayers - Returns the players who are AP-Null players.

A_DummyPlayer_propertyQ - Checks if the solution x satisfies the A-Dummy player property.

A_NullPlayer_propertyQ - Checks if the solution x satisfies the A-Null player property.

A_NullPlayers - Returns the players who are A-Null players.

AdditiveQ - Checks if the game v is additive.

AllMarginalContributions - Computes all marginal contributions of a Tu game.

AllMarginalContributionsResToS - Computes all marginal worth vectors of a TU-game v restricted to coalition S.

AllSubGames - Computes all subgames.

AlmostAverageConcaveQ - Returns true whenever the game v is almost average concave.

AlmostAverageConvexQ - Returns true whenever the game v is almost average convex.

AlmostConcave_gameQ - Returns true whenever the game v is almost concave.

AlmostConvex_gameQ - Returns true whenever the game v is almost convex.

AntiCoreCoverQ - Checks if the anti-core cover is non-empty.

AntiCorePlot - Plots the anti-core.

AntiCoreVertices - Evaluates the vertices of the anti-core.

AntiImputationVertices - Computes all vertices of the anti imputation set.

AntiReduced_game_propertyQ - Checks whether an imputation x satisfies the anti-reduced game property.

AntiUtopiaPayoff - Computes the anti-utopia and agreement vector.

Anti_B0_balancedCollectionQ - Checks the reversal of weak Kohlberg's criterion.

Anti_BestCoalitions - Computes the set of less effective coalitions.

Anti_CPCore - Computes the closest point of the anti-core to x.

Anti_ChiValue - Computes the anti-chi-value of a TU-game v.

Anti_Converse_DGP_Q - Checks whether an imputation x satisfies the anti-converse derived game property.

Anti_DerivedGame - Computes from (v,x,S) a modified Davis-Maschler anti-derived game vS on S at x for game v.

Anti_Derived_game_propertyQ - Checks whether an imputation x satisfies a modified anti-derived game property.

Anti_GenGap - Computes the anti-generalized gap function from game v.

Anti_Kernel - Computes an anti-kernel point.

Anti_LorenzDom - Checks if x anti-Lorenz dominates y in game v.

Anti_LorenzMinACoreQ - Checks if x is Anti_Lorenz minimal in the anti-core of game v, i.e., x is in the Anti_Lorenz set.

Anti_LorenzSol - Determines the anti-Lorenz solution of game v.

Anti_ModPreKernel - Computes from (v,x) an anti-modified pre-kernel element.

Anti_ModPreKernel - Computes from (v,x) an anti-modified pre-kernel element.

Anti_ModPreKernel2 - Computes from (v,x) an anti-modified pre-kernel element.

Anti_ModPrekernelQ - Checks whether the imputation x is a modified anti-pre-kernel element of the TU-game v.

Anti_Modiclus - Computes the anti modiclus of a game.

Anti_Monotonic_Cover - Computes the anti-monotonic cover of game v.

Anti_Nucl - Computes the anti nucleolus of a game.

Anti_Nucl_llp - Computes the anti nucleolus of a game.

Anti_PModPreKernel - Computes from (v,x) an anti-proper-modified pre-kernel element.

Anti_PModPrekernelQ - Checks whether the imputation x is a proper modified anti-pre-kernel element of the TU-game v.

Anti_PreKernel - Computes an anti-prekernel point.

Anti_PreNucl - Computes the anti pre-nucleolus of game v.

Anti_PreNucl_llp - Computes the anti pre-nucleolus of game v.

Anti_PrekernelQ - Checks if an imputation is an anti prekernel point.

Anti_PropModPreKernel - Checks whether the imputation x is a proper modified anti-pre-kernel element of the TU-game v.

Anti_TauValue - Computes the anti-tau-value of a TU-game v.

Anti_Weak_balancedCollectionQ - Verifies whether the set of induced coalitions is a weak_balanced collection.

Anti_balancedCollectionQ - Checks the reversal of Kohlberg's criterion.

Anti_kernelQ - Checks if an imputation is an anti kernel point.

Anti_modiclusQ - Verifies whether the set of induced coalitions is a bi-balanced collection.

B0_balancedCollectionQ - Checking weak Kohlberg's criterion.

B0_balancedQ - Verifies whether the collection of coalitions is weakly balanced.

BanzhafColeman - Computes the Banzhaf/Coleman index of a simple game sv (normalized Banzhaf value by 2^n-1).

BanzhafOwenValue - Computes the Banzhaf-Owen value w.r.t. a priori unions cs.

BanzhafPenrose - Computes the Banzhaf/Penrose and Banzhaf/Coleman index of a simple game sv.

BaryCenter - Computes the barycenter of the core.

BestCoalitions - Computes the set of most effective coalitions.

COV_propertyQ - Verifies if the payoff x satisfies COV property.

CPCore - Computes the closest point of the core to x.

CanonicalOrder - Orders a set of coalitions by a canonical order.

CddAntiCoreCoverPlot - Plots the anti-core cover set.

CddAntiCoreCoverVertices - Computes all vertices of the anti-core cover set.

CddAntiCorePlot - Plots the anti-core of a game using cddmex.

CddAntiCoreQ - Checks if the anti-core exists (cddmex).

CddAntiCoreSimplexPlot - Plots the anti-core using simplex projection.

CddAntiCoreSimplexVertices - Computes all anti-core vertices using simplex projection.

CddAntiCoreVertices - Computes the vertices of the anti-core (cddmex).

CddAntiImputationSimplexVertices - Computes all vertices of the anti-imputation set using simplex projection.

CddAntiImputationVertices - Computes all vertices of the anti-imputation set.

CddAntiLeastCore - Computes the least core of game v using (cddmex).

CddAntiLeastCoreVertices - Computes the vertices of the anti least core of game v (cddmex).

CddAntiNucl - Computes the anti nucleolus of game v (cddmex).

CddAntiNucl_llp - Computes the anti nucleolus of game v (cddmex).

CddAntiPrenucl - Computes the anti pre-nucleolus of game v (cddmex).

CddAntiPrenucl_llp - Computes the anti pre-nucleolus of game v (cddmex).

CddAnti_WeberSetPlot - Plots the anti-Weber set of game v.

CddAnti_WeberSetSimplexPlot - Plots the anti-Weber set of game v projected to the simplex.

CddBelongToLeastCoreQ - Checks if a payoff vector belongs to the least-core.

CddCoreCoverPlot - Plots the core cover of a TU game.

CddCoreCoverSimplexPlot - Plots the core cover (simplex projection).

CddCoreCoverSimplexVertices - Computes all vertices of the core cover (simplex).

CddCoreCoverVertices - Computes all vertices of the core cover of a TU game.

CddCoreMovie - Creates a movie w.r.t. the strong epsilon-cores.

CddCorePlot - Plots the core of a game using cddmex.

CddCoreQ - Checks if the core exists (cddmex).

CddCoreSimplexMovie - Creates a movie w.r.t. the strong epsilon-cores (simplex projection).

CddCoreSimplexPlot - Plots the core (simplex projection).

CddCoreSimplexVertices - Computes the vertices of the core (simplex).

CddCoreVertices - Computes the vertices of the core (cddmex).

CddExactGame - Computes the exact game from v (cddmex).

CddImputationSimplexVertices - Computes the vertices of the imputation set (simplex).

CddImputationVertices - Computes the vertices of the imputation set (cddmex).

CddKernelCatchers - Draws some kernel catchers (cddmex).

CddKernelCatchersSimplex - Draws some kernel catchers (simplex).

CddLeastCore - Computes the least core (cddmex).

CddLeastCoreVertices - Computes the least core vertices (cddmex).

CddLinear_Production - Computes from a production problem (A,mB,p) a linear production game using cddmex.

CddLowerSetSimplexVertices - Computes the vertices of the lower set (simplex).

CddLowerSetVertices - Computes the vertices of the lower set (cddmex).

CddModiclus - Computes the modiclus of game v using cddmex.

CddNucl - Computes the nucleolus using the CDD solver (cddmex).

CddPreKernel - Computes a pre-kernel element (cddmex).

CddPrenucl - Computes the prenucleolus using the CDD solver (cddmex).

CddPrenucl_llp - Computes the prenucleolus using the CDD solver (cddmex).

CddReasonableSetSimplexVertices - Computes the vertices of the reasonable set (simplex).

CddReasonableSetVertices - Computes the vertices of the reasonable set (cddmex).

CddStrongCorePlot - Plots a strong epsilon core.

CddStrongCoreSimplexPlot - Plots the strong epsilon core (simplex projection).

CddTotallyBalancedQ - Checks whether the core of all subgames is non-empty (cddmex).

CddUpperSetSimplexVertices - Computes the vertices of the upper set (simplex).

CddUpperSetVertices - Computes the vertices of the upper set (cddmex).

CddWeberSet - Computes the vertices of the Weber Set.

CddWeberSetPlot - Plots the Weber set.

CddWeberSetSimplex - Computes the vertices of the Weber Set (simplex).

CddWeberSetSimplexPlot - Plots the Weber set (simplex).

ChebyCenterCore - Computes the Cheby Center of the core of game v using MPT3.

ChiValue - Computes the chi-value of a TU-game v. This is a generalized Tau value.

CmpConsistencyQ - Checks whether an imputation x satisfies the complement consistency.

CmpRedGame - Computes from (v,x,S) a complement reduced game vS on S at x for game v.

CoalitionSolidarity - Determines the coalition solidarity value.

ColemanOwenValue - Computes the Coleman-Owen value w.r.t. a priori unions cs.

Complement_Reduced_game - Computes from (v,x) all complement reduced games on S at x of game v.

ComposeMarkets - Composite from a parameter set defining number of players involved in each separated market situation a market game (merged markets).

Composition - Composite at least 2 TU games up to 15 to a new extended game.

ConstantSumQ - Checks if the game v has constant-sum.

Converse_CmpConsistencyQ - Checks whether an imputation x satisfies the converse complement consistency property.

Converse_DGP_Q - Checks whether an imputation x satisfies the converse derived game property.

Converse_RGP_Q - Checks if an imputation satisfies the CRGP.

CoreCoverQ - Checks if the core cover a TU game v is non-empty.

CorePlot - Plots the core.

CoreVertices - Computes the vertices of the core.

Cost_Nucl - Computes the nucleolus of a cost game v using the optimization toolbox.

Cost_PreNucl - Computes the pre-nucleolus of a cost game v using the optimization toolbox.

DCP_propertyQ - Checks whether the solution x satisfies the dual cover property.

DFP_propertyQ - Checks whether the solution x satisfies the dual floor property.

DM_AntiReduced_game - Computes from (v,x) all anti-reduced games on S at x of game v.

DM_Anti_Derived_game - Computes from (v,x) a modified Davis-Maschler anti-reduced game vS on S at x for game v.

DM_Derived_game - Computes from (v,x) a modified Davis-Maschler reduced game vS on S at x for game v.

DM_Reduced_game - Computes all Davis-Maschler reduced games.

DM_TwoPersonGame - Computes from (v,x) all reduced two-person games.

DM_TwoReduced_game - Computes from (v,x) all single and two-person reduced games on S at x of game v.

DRP_propertyQ - Checks whether the solution x satisfies the dual replication property.

DecomposableQ - Checks whether the game v is decomposable w.r.t. the coalition structure cs.

DecomposeGame - Computes the unique decomposition of a TU-game.

DecomposeInPositiveGames - Decomposes a TU game v into the difference of two positive games (convex games).

DeeganPackel - Computes the Deegan-Packel index from the set of minimal winning coalitions.

DeeganPackel_SV - Computes the Deegan-Packel index from a simple game to construct the set of minimal winning coalitions.

DerivedCostMatrix - Computes from a cost matrix and a partition of the player set N the corresponding derived cost matrix.

DerivedGame - Computes from (v,x,S) a modified Davis-Maschler derived game vS on S at x for game v.

Derived_game_propertyQ - Checks whether an imputation x satisfies a modified derived game property.

DiscShapleyValue - Computes the discounted Shapley value.

DualCover - Computes the maximum characteristic values from the primal or dual game.

DualFloor - Computes the minimum characteristic values from the primal or dual game.

Dual_Cover_game - Computes from (v,x) a modified Davis-Maschler reduced game vS on S at x for game v.

Dual_Cover_propertyQ - Checks whether an imputation x satisfies a modified reduced game property

Dual_Floor_game - Computes from (v,x) a modified Davis-Maschler anti-reduced game vS on S at x for game v.

Dual_Floor_propertyQ - Checks whether an imputation x satisfies a modified anti-reduced game property.

DummyPlayer_propertyQ - Checks the dummy player property.

DummyPlayers - Returns the list of dummy players of game v.

DuttaRay - Computes the Dutta-Ray solution for convex games.

EANSCValue - Computes the Equal Allocation of Non-Separable Contribution/Cost Value.

ECCoverGame - Computes from (v,x) an excess comparability cover of game v.

ECFloorGame - Computes from (v,x) an excess comparability floor of game v.

ECGValue - Computes the Equal Collective Gains value of a TU-game v.

EC_DGP_Q - Checks whether the solution x satisfies excess comparability for each derived game.

EC_RGP_Q - Checks whether the solution x satisfies excess comparability for each reduced game.

EC_propertyQ - Checks whether the solution x satisfies excess comparability.

EPSDValue - Computes the egalitarian proportional surplus division value of a individually positive TU-game.

ESD - Computes the equal surplus division of a TU-game.

EqDistDividends - Computes the equally distributed dividends of coalitions to which players belong, i.e., Shapley-value, of a TU-game v.

EssentialConstSumQ - Checks if v is an essential constant-sum game.

EssentialQ - Checks if the game v is essential.

ExtShapleyValue - Computes the extended Shapley-value.

FindPartition - Tries to find a partition and anti partition w.r.t. payoff vector x.

FlatQ - Checks if the game v is flat.

Gap - Determines the gap function.

GatelyValue - Computes the Gately point of an essential game v.

GenGap - Computes the generalized gap function from game v.

GetAirPortProb - Computes a pseudo-random airport cost allocation problem of size m+1.

GetMCNetRules - Transforms a cell array of the MC-nets representation of a TU game into a structure array.

GetMarketGame - Determines from a random generated game the corresponding market game.

GetPlayersCharacter - Determines from the set of players of the weighted majority game the characters Step, Sum and Null Player.

GetPlayersCharacter - Determines from the set of players of the weighted majority game the characters Step, Sum and Null Player.

GetProbDist - Tries to find for the game v the corresponding probability distribution over the players' orderings w.r.t. pre-imputation x.

HMS_AntiReduced_game - Computes from (v,x) all Hart/Mas-Colell anti-reduced games on S at x of game v.

HMS_Anti_Derived_game - Computes from (v,x,S) a modified Hart-Mas-Colell anti-reduced game vS on S at x for game v.

HMS_Derived_game - Computes from (v,x,S) a modified Hart-Mas-Colell reduced game vS on S at x for game v.

HMS_DervGame - Computes from (v,x,S) a modified Hart-Mas-Colell derived game vS on S at x for game v.

HMS_ImputSavingReducedGame - Computes from (v,x) all Hart/Mas-Colell ISR games.

HMS_RedGame - Computes from (v,x,S) a Hart-Mas-Colell reduced game vS on S at x for game v.

HMS_Reduced_game - Creates all Hart/Mas-Colell reduced games.

HMS_TwoReduced_game - Computes from (v,x) all Hart/Mas-Colell singleton and two-person reduced games on S at x of game v.

HarsanyiValue - Computes a Harsanyi-value of a TU-game v.

ISRG_propertyQ - Checks whether an imputation x satisfies the ISR game property.

ImpSetEqsLwsQ - Checks if the imputation set coincides with the lower set.

ImputSavingReducedGame - Computes from (v,x) all imputation saving reduced games.

ImputationVertices - Computes the vertices of the imputation set.

InessGame - Computes the inessential game from a payoff vector.

InteractionSets - Determines a system of interaction sets.

IrredAntiCore - Computes from a cost matrix the corresponding extreme points of the irreducible anti-core of the associated m.c.s.t. game.

IrredCostMatrix - Computes from a cost matrix and a solution tree the irreducible cost matrix.

Johnston - Computes the Johnston power index from the set of winning coalitions.

Kernel - Computes a kernel point using optimization toolbox.

KrEqsPrkQ - Checks if the kernel is equal to the pre-kernel.

LED - Computes the large excess difference w.r.t. the payoff x.

LED_propertyQ - Checks whether the solution x satisfies large excess difference property.

LS_Nucl - Computes the least square nucleolus of a game.

LS_PreNucl - Computes the least square pre-nucleolus of a game.

LeastCore - Computes the least core using optimization toolbox.

LeastCoreVertices - Computes the least core vertices.

LedcoconsQ - Checks whether an imputation x satisfies large excess difference converse consistency.

Ledcons_propertyQ - Checks whether an imputation x satisfies the ledcons property

LexMaxMinWin - Determines the lexicographical maximal coalition from the clm represented as a cell, matrix, or array of integers.

LexOrder - Sorts a set of coalitions represented as a cell, matrix, or array of integers into the corresponding lexicographical order.

LorenzDom - Checks if x Lorenz dominates y in game v.

LorenzMaxCoreQ - Checks if x is Lorenz maximal in the core of game v, i.e., x is in the Lorenz set.

LorenzSet - Determines the Lorenz set of game v.

LorenzSol - Determines the Lorenz solution of game v.

MIMC - Computes the vector of minimum increase in players marginal contribution when they leave the grand coalition.

MLextension - Computes the multi-linear extension.

MMExcess - Computes the minimal and maximal excess vector of game v and its dual.

MTRCostMatrix - Computes from a cost matrix and a solution tree the cost matrix of a minimal spanning tree.

MarketGameQ - Checks whether the game v is a market game.

MaxConsistencyQ - Checks whether an imputation x satisfies maximal consistency.

MinimalRep - Computes from a simple game v and a threshold th of the minimal representation of an homogeneous weighted majority game.

ModDeeganPackel - Computes the modified Deegan-Packel index from the set of winning coalitions.

ModDeeganPackel_SV - Computes the Deegan-Packel index from a simple game to construct the set of minimal winning coalitions.

ModHoller - Computes a modified Holler index from the set of winning coalitions.

ModPGI - Computes the modified public good index from the set of minimal winning coalitions.

ModPGI_SV - Computes the modified public good index from a simple game to determine the set of minimal winning coalitions.

ModPreKernel - Computes from (v,x) a modified pre-kernel element.

ModPrekernelQ - Checks whether the imputation x is a modified pre-kernel element of the TU-game v.

Modiclus - Computes the modiclus of a game.

MyersonValue - Computes the Myerson value of a Tu game.

NetworkBanzhaf - Computes the network Banzhaf power index from the set of winning coalitions of a network E while imposing a threshold of th.

NetworkCenterSol - Computes the center solution from the minimal representation of a homogeneous network weighted majority game.

NetworkDeeganPackel - Computes the network Deegan-Packel index from the set of winning coalitions of a network E while imposing a threshold of th.

NetworkJohnston - Computes the network Johnston power index from the set of winning coalitions of a network E while imposing a threshold of th.

NetworkMajorityGame - Computes a network majority TU game (simple game).

NetworkMinimalRep - Computes from the set of edges, threshold th and the weights w_vec the minimal homogeneous representation of a homogeneous network weighted majority game.

NetworkModDeeganPackel - Computes the network modified Deegan-Packel index from the set of winning coalitions of a network E while imposing a threshold of th.

NetworkModPGI - Computes the modified network public good index from the set of minimal winning coalitions of a network.

NetworkPGI - Computes the network public good index from the set of minimal winning coalitions of a network.

NetworkShapleyShubik - Computes the network Shapley-Shubik power index from the set of winning coalitions of a network E while imposing a threshold of th.

NucAirportProb - Computes the nucleolus from an airport capital cost problem.

NullPlayer_propertyQ - Verifies if x satisfies the null player property.

NullPlayers - Returns the list of null players of game v.

One_Normalization - Computes from the game v the corresponding one-normalized game.

OwenValue - Computes the Owen value.

PDValue - Computes the proportional division value of a individually positive TU-game.

PGI - Computes the public good index from the set of minimal winning coalitions.

PGI_SV - Computes the public good index from a simple game to determine the set of minimal winning coalitions.

PModPreKernel - Computes from (v,x) a proper modified pre-kernel element.

PModPrekernelQ - Checks whether the imputation x is a proper modified pre-kernel element of the TU-game v.

PRP_propertyQ - Checks whether the solution x satisfies the primal replication property.

PS_GameBasis - Computes the basis for the class of PS games.

PartitionGameQ - Verifies if (th,w_vec) induces a partition game.

PartitionPlySet - Partitions the set of players of the weighted majority game into character Sum, Step, and Null-Player.

PartitionSA - Computes a partition of S w.r.t. a hypergraph communication situation.

PartitionSL - Computes a partition of S w.r.t. a communication situation.

PermutationGame - Computes from an assignment matrix the permutation game.

PlayersCharacter - Partitions the set of players of the weighted majority game into the character Sum, Step, and Null-Player.

PlotCostGraph - Plots from a cost matrix the associated cost spanning graph.

PlyEqFirstStep - Returns the last player that is equivalent to the first step.

PositionValue - Computes the position value.

Potential - Determines the potential of a TU game (recursive).

PowerSet - Computes all subsets from a set representation.

PreKernel - Computes a prekernel element.

PreNucl - Computes the prenucleolus using optimization toolbox.

PreNucl2 - Computes the prenucleolus using optimization toolbox.

PreNucl_llp - Computes the prenucleolus using optimization toolbox.

PreNucl_mod - Computes the pre-nucleolus of game v using the optimization toolbox and a spanning method.

PreNucl_mod2 - Computes the pre-nucleolus of game v using the optimization toolbox and an alternative spanning method.

PreNucl_mod3 - Computes the pre-nucleolus of game v using the optimization toolbox and an alternative spanning method.

PreNucl_mod4 - Computes the pre-nucleolus of game v using the optimization toolbox and an alternative spanning method.

PrekernelQ - Checks if an imputation is a pre-kernel point.

PrenuclQ - Checks if an imputation is the pre-nucleolus using Kohlberg's criterion.

PrkEqPnQ - Checks whether the pre-kernel of a game v is a singleton.

PrkEqsModPrkQ - Checks whether a pre-kernel element is also an element of the modified as well as proper modified pre-kernel

PropModPreKernel - Computes from (v,x) a proper modified pre-kernel element from the dual cover game.

PropNucl - Computes the proportional nucleolus.

PropPreNucl - Computes the proportional pre-nucleolus.

ProportionalGame - Computes from (q,co) a proportional game.

REAS_LED_DCGame - Verifies that x is a reasonable vector of game v, then the shifted ducal cover game satisfies LED w.r.t. the replicated vector (x,x).

REAS_propertyQ - Checks if the vector x satisfies the reasonableness on both sides

REC_propertyQ - Checks whether the solution x satisfies reverse excess comparability.

RE_RGP - Checks whether an imputation x is reasonable from both sides for all reduced games.

ReasSetEqsUpsQ - Checks if the reasonable set coincides with the upper set.

Reconfirmation_propertyQ - Checks the RCP.

RedGame - Creates a Davis-Maschler reduced game.

Reduced_game_propertyQ - Checks the RGP.

ReverseMCNetsRep - Reverse the MC-nets (concise) representation of a cooperative game into a full representation.

SDCP_propertyQ - Checks whether the solution x satisfies a strong dual cover property.

SDFP_propertyQ - Checks whether the solution x satisfies a strong dual floor property.

SD_ShapleyValue - Computes the surplus division Shapley value.

SED - Computes the small excess difference w.r.t. the payoff x.

SED_propertyQ - Checks whether the solution x satisfies small excess difference property.

SedcoconsQ - Checks whether an imputation x satisfies small excess difference converse consistency.

Sedcons_propertyQ - Checks whether an imputation x satisfies the sedcons property.

ShapAirPortMod2 - Computes the Shapley value from an airport capital cost problem (modified).

ShapleyAirportProb - Computes the Shapley value from an airport capital cost problem.

ShapleyQ - Checks if the imputation x is a Shapley value of game v.

ShapleyValue - Computes the Shapley value (potential).

ShapleyValueLB - Computes the Shapley value from the linear basis.

ShapleyValueM - Computes the Shapley value based on all marginal contributions.

ShapleyValueML - Computes the Shapley value using multi-linear extension.

ShapleyValuePot - Computes the Shapley value and potential.

SolidarityPGI - Computes the solidarity Holler index w.r.t. a priori unions cs.

SolidarityShapleyValue - Determines the solidarity Shapley value.

SolidarityValue - Determines the solidarity value.

SortMg - Sorts a sub/power set w.r.t. its cardinality.

SplitSimpleGame - Splits a simple game with winning players into sub-games with and without winning players.

StandardSolution - Determines the standard solution.

StrConverse_DGP_Q - Checks whether an imputation x satisfies the strong converse derived game property.

StrConverse_RGP_Q - Checks the strong RGP.

StrLedcoconsQ - Checks whether an imputation x satisfies satisfies strong large excess difference converse consistency.

StrSedcoconsQ - Checks whether an imputation x satisfies satisfies strong small excess difference converse consistency (SEDCOCONS).

SubCoalitions - Computes the power set (subsets) from an array.

SubDual - Determines the dual of a subgame.

SubGame - Creates a subgame.

SubSets - Creates all subsets of super set.

Sum_Marg_Contributions - Returns 1 whenever for a coalition the sum of marginal contributions is positive.

SuperAddSolQ - Checks if the vector x is an element of a super additive solution of the game v.

SuperSets - Computes the super-sets of set S.

Talmudic_Rule - Computes the Talmudic rule.

TauValAirportProb - Computes the tau value from an airport capital cost problem.

TauValue - Computes the Tau value.

UnionStableBasis - Determines a basis of a union stable system.

UpperPayoff - Computes the upper and minimum claim vector of game v.

WSysKernelQ - Checks the correctness of a kernel vector w.r.t. a weight system.

WSysNuclQ - Checks the correctness of the nucleolus w.r.t. a weight system.

WSysPreKernelQ - Checks the correctness of a pre-kernel vector w.r.t. a weight system.

WSysPreNuclQ - Checks the correctness of the pre-nucleolus w.r.t. a weight system.

WeakReduced_game_propertyQ - Checks whether an imputation satisfies the weak reduced game property.

Weak_balancedCollectionQ - Checking weak Kohlberg's criterion.

Weak_balancedCollectionQ - Checking weak Kohlberg's criterion.

WedgeProdGame - Computes from an MC-nets (concise) representation a cooperative production game (wedge production game).

ZeroOne_Normalization - Creates a zero-one normalized game.

additive_game - Creates an additive game.

admissibleGame - Computes a symmetric compromise admissible game.

airport_costgame - Computes from an airport problem the associated airport cost game.

airport_game - Computes from an airport problem the associated savings game.

airport_profit - Computes from a cost and benefit vector the associated surplus game.

alphaVector - Computes recursively an alpha vector from the positive core.

anti_coreQ - Checks the existence of the anti-core of game v.

anti_partition - Computes from a partition its anti partition.

apex_game - Creates an apex game.

apu_PGI - Computes the Holler index w.r.t. a priori unions cs.

apu_SolidarityValue - Determines the solidarity value w.r.t. a priori unions.

assignment_game - Creates an assignment game.

average_concaveQ - Returns true whenever the game v is average-concave.

average_convexQ - Checks the Tu-game on average convexity.

average_excess - Computes the average excess of game v.

balancedCollectionQ - Checking Kohlberg's criterion.

balancedCoverQ - Checks whether the characteristic function v is equal to a balanced cover.

balancedQ - Verifies whether the collection of coalitions is balanced.

bankruptcy_airport - Computes from a bankruptcy problem the airport surplus game.

bankruptcy_game - Creates a bankruptcy game.

banzhaf - Computes the Banzhaf value.

basis_coordinates - Determines the basis coordinates of a Tu game.

basis_game - Determines bases games.

belongToAllSubCores - Checks whether all projections of an imputation x are a member of an associated core of a subgame.

belongToAntiCoreQ - Checks if a payoff vector belongs to the anti-core.

belongToCoreQ - Checks if a payoff vector belongs to the core.

belongToImputationSetQ - Checks if a payoff vector belongs to imputation set.

belongToLeastCoreQ - Checks if a payoff vector belongs to the least core.

belongToLowerSetQ - Checks if a payoff vector belongs to lower set.

belongToUpperSetQ - Checks if a payoff vector belongs to upper set.

belongToWeberSetQ - Checks if the imputation x belongs to the Weber set using MPT3.

bidding_collusion_game - Determines a bidding collusion game from a bid vector.

bint_AssignmentGame - Creates an assignment game (bintprog).

bs_PreKernel - Computes from (v,x) a pre-kernel element using MATLAB's backslash instead of pinv.

cardinality_game - Assigns zero to a coalition of size<=k<n, otherwise its cardinality.

cardinality_game2 - Assigns a zero to a coalition of size<=k<n otherwise its cardinality times 100.

center_solution - Computes the center solution from the minimal representation of a homogeneous weighted majority game.

cfr_Anti_Kernel - Computes an anti kernel element with coalition formation restrictions.

cfr_Anti_Nucl - Computes the anti nucleolus of game v with coalition formation restrictions.

cfr_Anti_PreKernel - Computes an anti-pre-kernel element with coalition formation restrictions.

cfr_Anti_PreNucl - Computes the anti pre-nucleolus of game v with coalition formation restrictions.

cfr_Anti_Weak_balancedCollectionQ - Verifies whether the set of induced coalitions is a weak_balanced collection with coalition formation restrictions.

cfr_Anti_balancedCollectionQ - Verifies whether the set of induced coalitions is an anti balanced collection with coalition formation restrictions.

cfr_Kernel - Computes a Kernel element with coalition formation restrictions.

cfr_PreKernel - Computes a Pre-Kernel element with coalition formation restrictions.

cfr_PreNucl - Computes the pre-nucleolus of game v with coalition formation restrictions.

cfr_balancedCollectionQ - Verifies whether the set of induced coalitions is a balanced collection with coalition formation restrictions.

cfr_nucl - Computes the nucleolus of game v with coalition formation restrictions.

cfr_weak_balancedCollectionQ - Verifies whether the set of induced coalitions is a weakly_balanced collection with coalition formation restriction.

clToMatlab - Computes the unique integer representation of coalitions.

clp_kernel - Computes a kernel point using the CLP solver.

clp_weightedKernel - Computes a weighted kernel point using the CLP solver.

cls_kernel - Computes a kernel point using the CLS solver.

cls_weightedKernel - Computes a weighted kernel point using the CLS solver.

coeff_linearbasis - Determines the coefficients (dividends) of a linear basis from a TU game.

complementary_basis - Computes the complementary basis of the n-person TU game space.

complementary_dividends - Computes the complementary dividends.

complementary_game - Generates a producer and buyer game.

compromiseAdmissibleQ - Checks if the core cover a TU game v is non-empty.

compromiseAntiAdmissibleQ - Checks if the anti-core cover a TU game v is non-empty.

compromiseStableQ - Checks if the game is compromise stable.

concave_gameQ - Checks the concavity of a Tu-game.

contentment - Computes the contentment vector of game v w.r.t. x.

convex_gameQ - Checks the convexity of a Tu-game.

coreQ - Checks the non-emptiness of the core.

cp_kernel - Computes a kernel element using the cone programming solver of MATLAB's Optimization Toolbox.

cp_prekernel - Computes a pre-kernel element using the cone programming solver of MATLAB's Optimization Toolbox.

cplex_AntiNucl - Computes the anti nucleolus of game v using the CPLEX solver.

cplex_AntiNucl_llp - Computes the anti nucleolus of game v using the CPLEX solver.

cplex_AntiPreNucl - Computes the anti prenucleolus using the CPLEX solver.

cplex_AntiPreNucl_llp - Computes the anti prenucleolus using the CPLEX solver.

cplex_AntiPreNucl_mod - Computes the anti pre-nucleolus of game v using cplexmex (fast).

cplex_AntiPreNucl_mod2 - Computes the anti pre-nucleolus of game v using cplexmex (fast/method 2).

cplex_AntiPreNucl_mod3 - Computes the anti pre-nucleolus of game v using cplexmex (fast/method 3).

cplex_AntiPreNucl_mod4 - Computes the anti pre-nucleolus of game v using cplexmex (fast/method 4).

cplex_AssignmentGame - Creates an assignment game using the CPLEX solver.

cplex_LeastCore - Computes the least core using cplexmex.

cplex_alphaVector - Computes recursively an alpha vector from the positive core of game v using cplexmex.

cplex_cfr_nucl - Computes the nucleolus of game v with coalition formation restrictions using the CPLEX solver.

cplex_cs_nucl - Computes the nucleolus of game v w.r.t. coalition structure cs using the CPLEX solver.

cplex_exact_game - Computes the exact game from v using the CPLEX solver.

cplex_flow_game - Computes from a flow problem a TU flow game (CPLEX).

cplex_kernel - Computes a kernel point using the CPLEX solver.

cplex_linprog_game - Computes from a production matrix (A,H,mB,mD,p) a linear programming game using cplexmex.

cplex_modiclus - Computes the modiclus of game v using cplexmex.

cplex_nucl - Computes the nucleolus using the CPLEX solver.

cplex_nucl_llp - Computes the nucleolus using the CPLEX solver.

cplex_prekernel - Computes a prekernel point using the CPLEX solver.

cplex_prenucl - Computes the prenucleolus using the CPLEX solver.

cplex_prenucl_llp - Computes the prenucleolus using the CPLEX solver.

cplex_prenucl_mod4 - Computes the pre-nucleolus of game v using cplexmex (fast/method 4).

cplex_weightedKernel - Computes a weighted kernel point using the CPLEX solver.

cplex_weightedNucl - Computes a weighted nucleolus using the CPLEX solver.

cplex_weightedNucl_llp - Computes a weighted nucleolus using the CPLEX solver.

cplex_weightedPreKernel - Computes a weighted prekernel point using the CPLEX solver.

cplex_weightedPreNucl - Computes a weighted prenucleolus using the CPLEX solver.

cplex_weightedPreNucl_llp - Computes a weighted prenucleolus using the CPLEX solver.

critical_value1 - Computes the biggest gain of any group of players.

critical_value2 - Computes a critical value w.r.t. the strong epsilon-core.

critical_value_star - Computes a critical value which contains the intersection of the imputation and reasonable set

cs_Anti_Kernel - Computes an anti-kernel element from a coalition structure.

cs_Anti_Nucl - Computes the anti nucleolus of game v w.r.t. coalition structure cs.

cs_Anti_PreKernel - Computes an anti-pre-kernel element from a coalition structure.

cs_Anti_PreNucl - Computes the anti pre-nucleolus of game v w.r.t. coalition structure cs.

cs_Anti_Weak_balancedCollectionQ - Verifies whether the set of induced coalitions is a weak_balanced collection w.r.t. the coalition structure cs. Checking Kohlberg's criterion.

cs_Anti_balancedCollectionQ - Verifies whether the set of induced coalitions is an anti balanced collection w.r.t. the coalition structure cs. Checking Kohlberg's criterion.

cs_Banzhaf - Computes the Banzhaf value w.r.t. a communication situation.

cs_GetPrk - Computes a pre-kernel element from each possible partition of N.

cs_Kernel - Computes a kernel element from a coalition structure.

cs_PreKernel - Computes from (v,x) a pre-kernel element w.r.t. the coalition structure cs.

cs_PreNucl - Computes the pre-nucleolus of game v w.r.t. coalition structure cs.

cs_Weak_balancedCollectionQ - Verifies whether the set of induced coalitions is a weakly_balanced collection w.r.t. the coalition structure cs. Checking Kohlberg's criterion.

cs_balancedCollectionQ - Verifies whether the set of induced coalitions is a balanced collection w.r.t. the coalition structure cs. Checking Kohlberg's criterion.

cs_nucl - Computes the nucleolus of game v w.r.t. coalition structure cs.

cvx_kernel - Computes a kernel point using the CVX solver.

cvx_prekernel - Computes a prekernel point using the CVX solver.

cvx_weightedKernel - Computes a weighted kernel point using the CVX solver.

cvx_weightedPreKernel - Computes a weighted prekernel point using the CVX solver.

diffOperator - Computes the difference operator of the game w.r.t. coalition T.

disagreement - Computes the disagreement vector of game v.

dual_game - Creates the dual of a Tu-game.

equal_treatmentQ - Checks if a vector x satisfies ETP.

essentialSet - Computes the set of essential coalitions.

exact_game - Computes the exact game from v using Matlab's Optimization toolbox.

exact_gameQ - Checks whether game v is an exact game using Matlab's Optimization toolbox.

excess - Determines the excesses w.r.t. a payoff vector.

feasible_dividends - Computes a collection of feasible dividends.

flow_game - Computes from a flow problem a TU flow game using the optimization toolbox.

flow_probMinCut - Computes from a flow problem a minimal cut.

formatPowerSet - Formats the Matlab cell output that contains the representation of coalitions into matrix form.

gameToMama - Converts a TU-game into Mathematica representation.

gameToMatlab - Converts a Tu-game into Matlab representation.

game_Two - Constructs a 2-game from the coalition size 2 and number of players.

game_Wsys - Creates a set of games from an asymmetric weight system (all types).

game_basis - Computes a game basis of the n-person TU game space.

game_space - Computes the game space which replicates a payoff as a pre-kernel element.

genUnionStable - Creates a union stable system.

getCOV - Computes from a sample of observations obs and for n-assets the covariance matrix V of a portfolio with indefinite risk-return relationship.

getCOV2 - Computes from a sample of observations obs and for n-assets the covariance matrix V of a portfolio with negative risk-return relationship.

getCOV3 - Computes from a sample of observations obs and for n-assets the covariance matrix V of a portfolio with positive risk-return relationship.

getMinimalWinning - Computes from a simple game the minimal winning coalitions.

getPSgame - Computes a PS game from the PS game basis.

getSymCostMatrix - Computes a symmetric cost matrix from the cardinality of the player set and a upper bound value to specify the range from which the random number are drawn.

getgame - Creates a Tu-game from the unanimity coordinates.

glpk_AntiNucl - Computes the anti nucleolus of game v using the GLPK solver.

glpk_AntiNucl_llp - Computes the anti nucleolus of game v using the GLPK solver.

glpk_AntiPreNucl - Computes the anti pre-nucleolus using the GLPK solver.

glpk_AntiPreNucl_llp - Computes the anti pre-nucleolus using the GLPK solver.

glpk_alphaVector - Computes recursively an alpha vector from the positive core.

glpk_cfr_nucl - Computes the nucleolus of game v with coalition formation restrictions using the GLPK solver.

glpk_cs_nucl - Computes the nucleolus of game v w.r.t. coalition structure cs using the GLPK solver.

glpk_exact_game - Computes the exact game from v using the GLPK solver.

glpk_flow_game - Computes from a flow problem a TU flow game (GLPK).

glpk_kernel - Computes a kernel point using the GLPK solver.

glpk_linprog_game - Computes from a production matrix (A,H,mB,mD,p) a linear programming game using glpkmex.

glpk_modiclus - Computes the modiclus of game v using glpkmex.

glpk_nucl - Computes the nucleolus using the GLPK solver.

glpk_nucl_llp - Computes the nucleolus using the GLPK solver.

glpk_prekernel - Computes a prekernel point using the GLPK solver.

glpk_prenucl - Computes the prenucleolus using the GLPK solver.

glpk_prenucl_llp - Computes the prenucleolus using the GLPK solver.

glpk_weightedKernel - Computes a weighted kernel point using the GLPK solver.

glpk_weightedNucl - Computes a weighted nucleolus using the GLPK solver.

glpk_weightedNucl_llp - Computes a weighted nucleolus using the GLPK solver.

glpk_weightedPreKernel - Computes a weighted prekernel point using the GLPK solver.

glpk_weightedPreNucl - Computes a weighted prenucleolus using the GLPK solver.

glpk_weightedPreNucl_llp - Computes a weighted prenucleolus using the GLPK solver.

grMaxFlowGame - Computes from a flow problem a TU flow game.

greedy_bankruptcy - Creates the greedy bankruptcy game.

gurobi_AntiNucl - Computes the anti nucleolus of game v using the GUROBI solver.

gurobi_AntiNucl_llp - Computes the anti nucleolus of game v using the GUROBI solver.

gurobi_AntiPreNucl - Computes the anti prenucleolus using the GUROBI solver.

gurobi_AntiPreNucl_llp - Computes the anti prenucleolus using the GUROBI solver.

gurobi_AssignmentGame - Creates an assignment game using the GUROBI solver.

gurobi_alphaVector - Computes recursively an alpha vector from the positive core.

gurobi_cfr_nucl - Computes the nucleolus of game v with coalition formation restrictions using the GUROBI solver.

gurobi_cs_nucl - Computes the nucleolus of game v w.r.t. coalition structure cs using the GUROBI solver.

gurobi_exact_game - Computes the exact game from v using the GUROBI solver.

gurobi_flow_game - Computes from a flow problem a TU flow game (GUROBI).

gurobi_kernel - Computes a kernel point using the GUROBI solver.

gurobi_linprog_game - Computes from a production matrix (A,H,mB,mD,p) a linear programming game using gurobimex.

gurobi_modiclus - Computes the modiclus of game v using the GUROBI.

gurobi_nucl - Computes the nucleolus using the GUROBI solver.

gurobi_nucl_llp - Computes the nucleolus using the GUROBI solver.

gurobi_prekernel - Computes a prekernel point using the GUROBI solver.

gurobi_prenucl - Computes the prenucleolus using the GUROBI solver.

gurobi_prenucl_llp - Computes the prenucleolus using the GUROBI solver.

gurobi_weightedKernel - Computes a weighted kernel point using the GUROBI solver.

gurobi_weightedNucl - Computes a weighted nucleolus using the GUROBI solver.

gurobi_weightedNucl_llp - Computes a weighted nucleolus using the GUROBI solver.

gurobi_weightedPreKernel - Computes a weighted prekernel point using the GUROBI solver.

gurobi_weightedPreNucl - Computes a weighted prenucleolus using the GUROBI solver.

gurobi_weightedPreNucl_llp - Computes a weighted prenucleolus using the GUROBI solver.

harsanyi_dividends - Determines the the unanimity coordinates.

holler - Computes the Holler index.

homogeneous_representationQ - Checks if the weighted majority game possesses a homogeneous representation.

hsl_prekernel - Computes a prekernel point using HSL solvers.

hsl_weightedPreKernel - Computes a weighted prekernel point using HSL solvers.

hypergraphQ - Checks whether the system is a hypergraph communication situation.

interest_game - Computes from an interest problem the corresponding game.

intersection_basis - Computes the intersection basis of the n-person TU game space.

ipopt_kernel - Computes a kernel point using the IPOPT solver.

ipopt_prekernel - Computes a prekernel point using the IPOPT solver.

ipopt_weightedKernel - Computes a weighted kernel point using the IPOPT solver.

ipopt_weightedPreKernel - Computes a weighted prekernel point using the IPOPT solver.

ireffQ - Checks if a payoff satisfies IR as well as the Eff property.

jury_game - Computes from a quota and the number of jurors a simple game.

k_Converse_RGP_Q - Checks if an imputation satisfies the k-CRGP.

k_Reconfirmation_propertyQ - Checks the k-RCP.

k_Reduced_game_propertyQ - Checks the k-RGP.

k_StrConverse_RGP_Q - Checks the strong k-CRGP.

k_anticover - Determines from the Tu-game the corresponding anti k-game.

k_concaveQ - Checks k-concavity of the Tu-game.

k_convexQ	- Checks k-convexity of the Tu-game.
k_cover	- Determines from the Tu-game the corresponding k-game.
kernelQ	- Checks if an imputation is a kernel point.
landlord	- Computes a production game arising from l-landlords and t-tenants.
lin_prekernel	- Computes a prekernel point using optimization toolbox.
lin_weightedPreKernel	- Computes a weighted prekernel point using optimization toolbox.
linear_basis	- Determines the linear basis of the n-person TU game space.
linear_production	- Computes from a production problem (A,mB,p) a linear production game.
linprog_game	- Computes from a production matrix (A,H,mB,mD,p) a linear programming game.
lowersetQ	- Checks the existence of the lower set.
ma57_prekernel	- Computes from (v,x) a pre-kernel element using HSL MA57.
ma86_prekernel	- Computes from (v,x) a pre-kernel element using HSL MA86.
ma87_prekernel	- Computes from (v,x) a pre-kernel element using HSL MA87.
ma97_prekernel	- Computes from (v,x) a pre-kernel element using HSL MA97.
market2_game	- Determines from two disjoint sets a market game.
market_game	- Determines from two disjoint sets a market game.
mcst_game	- Computes from a cost matrix the corresponding mcst game.
mex_coalitions	- Computes the set of coalitions with maximum excesses
minNoBlockPayoff	- Computes the minimum no blocking payoff from game v.
min_aspiration	- Computes the minimum aspiration level of players of game v.
min_epsshift	- Computes for an almost-convex game the min epsilon shift to construct a convex game.
min_game	- Generates a minimum game.

min_homogrep - Computes from the threshold th and the weights w_vec the minimal homogeneous representation of an homogeneous weighted majority game.

minimal_representation - Computes from the threshold th and the weights w_vec the minimal representation of an homogeneous weighted majority game.

minimal_winning - Computes the minimal winning coalitions.

modiclusQ - Verifies whether the set of induced coalitions is a bi-balanced collection.

monotone_gameQ - Checks monotonicity of the TU game.

monotonic_cover - Determines the monotonic cover from a TU game.

msk_AntiNucl - Computes the anti nucleolus of game v using the MOSEK solver.

msk_AntiNucl_llp - Computes the anti nucleolus of game v using the MOSEK solver.

msk_AntiPreNucl - Computes the anti prenucleolus using the MOSEK solver.

msk_AntiPreNucl_llp - Computes the anti prenucleolus using the MOSEK solver.

msk_AssignmentGame - Creates an assignment game using the MOSEK solver.

msk_alphaVector - Computes recursively an alpha vector from the positive core.

msk_cfr_nucl - Computes the nucleolus of game v with coalition formation restrictions using the MOSEK solver.

msk_cs_nucl - Computes the nucleolus of game v w.r.t. coalition structure cs using the MOSEK solver.

msk_exact_game - Computes the exact game from v using the MOSEK solver.

msk_flow_game - Computes from a flow problem a TU flow game (MOSEK).

msk_kernel - Computes a kernel point using the MOSEK solver.

msk_linear_production - Computes from a production problem (A,mB,p) a linear production game using mosekmex.

msk_linprog_game - Computes from a production matrix (A,H,mB,mD,p) a linear programming game using mosekmex.

msk_modiclus - Computes the modiclus of game v using the MOSEK solver.

msk_nucl - Computes the nucleolus using the MOSEK solver.

msk_nucl_llp - Computes the nucleolus using the MOSEK solver.

msk_prekernel - Computes a prekernel point using the MOSEK solver.

msk_prenucl - Computes the prenucleolus using the MOSEK solver.

msk_prenucl_llp - Computes the prenucleolus using the MOSEK solver.

msk_prenucl_mod - Computes the pre-nucleolus of game v using mosekmex and a spanning method.

msk_prenucl_mod2 - Computes the pre-nucleolus of game v using mosekmex and an alternative spanning method.

msk_prenucl_mod3 - Computes the pre-nucleolus of game v using mosekmex and an alternative spanning method.

msk_prenucl_mod4 - Computes the pre-nucleolus of game v using mosekmex and an alternative spanning method.

msk_weightedKernel - Computes a weighted kernel point using the MOSEK solver.

msk_weightedNucl - Computes a weighted nucleolus using the MOSEK solver.

msk_weightedNucl_llp - Computes a weighted nucleolus using the MOSEK solver.

msk_weightedPreKernel - Computes a weighted prekernel point using the MOSEK solver.

msk_weightedPreNucl - Computes a weighted prenucleolus using the MOSEK solver.

msk_weightedPreNucl_llp - Computes a weighted prenucleolus using the MOSEK solver.

near_ringQ - Checks if a collection of coalitions is a near ring.

nucl - Computes the nucleolus using optimization toolbox.

nucl_formula - Computes the nucleolus of a three-person super-additive game v from a formula.

nucl_llp - Computes the nucleolus using optimization toolbox.

nullShapley - Determines a basis of the null space for the Shapley-value for n-persons.

nullShapleyLB - Determines a counting basis of the null space for the Shapley-value for n-persons.

oases_kernel - Computes a kernel point using the OASES solver.

oases_prekernel - Computes a prekernel point using the OASES solver.

oases_weightedKernel	- Computes a weighted kernel point using the OASES solver.
oases_weightedPreKernel	- Computes a weighted prekernel point using the OASES solver.
oddeven_game	- Assigns |S|-1 if S is odd and |S|+1 if S is even.
ols_prekernel	- Computes a prekernel point using optimization toolbox.
ols_weightedPreKernel	- Computes a weighted prekernel point using optimization toolbox.
one_concaveQ	- Checks whether the game v is 1-concave.
one_convexQ	- Checks whether the game v is 1-convex.
positive_gameQ	- Returns true (1) if all Harsanyi dividends are non-negative, otherwise false (zero).
potential	- Determines the potential of a TU game (basis).
probability_game	- Computes from a positive vector x the corresponding probability game.
product_game	- Computes form a vector x the corresponding product game.
production_game	- Creates an affine production game.
production_game2	- Creates an affine production game.
production_game_sq	- Creates a quadratic production game.
profit_matrix	- Creates the profit matrix of an assignment game.
proper_amount	- Computes the largest amount players contribute to a proper coalition.
ps_gameQ	- Checks whether a game is a PS game.
psstar_gameQ	- Checks whether a game is a PS* game.
pure_overhead	- Creates the matrix of pure overhead games.
qpBB_kernel	- Computes a kernel point using the QPBB solver.
qpBB_weightedKernel	- Computes a weighted kernel point using the QPBB solver.
qpc_kernel	- Computes a kernel point using the QPC solver.

qpc_prekernel - Computes a prekernel point using the QPC solver.

qpc_weightedKernel - Computes a weighted kernel point using the QPC solver.

qpc_weightedPreKernel - Computes a weighted prekernel point using the QPC solver.

qrg_prekernel - Computes a prekernel point using qrginv instead of pinv.

qrg_weightedPreKernel - Computes a weighted prekernel point using qrginv instead of pinv.

quotas - Determines the quotas of a game.

reasonable_outcome - Determines the reasonable outcome.

replicate_Shapley - Replicates the Shapley value for a game space.

replicate_prk - Replicates a pre-kernel solution as a pre-kernel of a game space.

root_game - Computes from game v its associated root game.

rootedQ - Checks whether the game v is rooted.

satisfaction - Computes the satisfaction of a partition and of its anti partition.

savings_game - Creates a saving game from a cost game.

scrb_solution - Computes separable costs-remaining benefits allocation.

secDiffOperatorQ - Checks whether the second order differences are all positive which indicates convexity.

select_starting_pt - Selects a starting point for the pre-kernel computation.

semi_concaveQ - Checks semi-concavity.

semi_convexQ - Checks semi-convexity.

separable_cost_allocation - Computes the separable cost allocation.

separating_collectionQ - Verifies if a collection is separating.

shiftGame - Computes from the game v the t-shift game of v.

simple_game - Creates a simple game.

sm_Kernel - Computes an element of the simplified Kernel of a game.

sm_PreKernel - Computes an element of the simplified pre-kernel of game v.

sm_PreNucl - Computes the simplified pre-nucleolus of a game.

sm_nucl - Computes the simplified nucleolus of a game.

smallest_amount - Computes the smallest amount vector.

sortsets - Sorts a sub/power set w.r.t. its cardinality.

streps_value - Determines the strong epsilon-game.

strictconvex_gameQ - Returns 1 whenever the game v is strictly convex.

sub_additiveQ - Returns true whenever the game v is sub-additive.

substitutes - Establishes which pair of players are substitutes.

super_additiveQ - Checks the Tu-game on super additivity.

superadditive_cover - Computes from game v its superadditive cover.

surplus_game - Computes from a cost game c the corresponding surplus game v.

symmetricQ - Checks if the game v is symmetric.

totallyAntiBalancedQ - Checks whether the anti-core of all subgames is non-empty.

totallyBalancedCoverQ - Checks whether the characteristic function v is equal to a totally balanced cover.

totallyBalancedQ - Checks whether the core of all subgames is non-empty.

tricameral_assembly - Computes from a set of parameters a simple game.

unanimity_games - Computes the unanimity coordinates.

union_stableQ - Checks whether a system is union stable.

uppersetQ - Checks the existence of the upper set.

value_matrix - Computes from an assignment matrix the corresponding value matrix for a permutation game.

vclToMatlab - Computes a Tu-game and the corresponding unique integer representation of coalitions

veto_players - Determines the veto players of a simple game.

veto_rich_players - Returns a list of veto players for the TU-game v.

weakly_sub_additiveQ - Returns true whenever the game v is weakly sub-additive.

weakly_super_additiveQ - Checks the Tu-game on weakly super additivity.

weightedAntiBalancedCollectionQ - Checks the reversal of weighted Kohlberg's criterion.

weightedAnti_B0_balancedCollectionQ - Checks the reversal of weighted weak Kohlberg's criterion.

weightedAnti_Kernel - Computes a weighted anti-kernel point.

weightedAnti_Nucl - Computes a weighted anti nucleolus of game.

weightedAnti_Nucl_llp - Computes a weighted anti nucleolus of game.

weightedAnti_PreKernel - Computes a weighted anti-prekernel point.

weightedAnti_PreNucl - Computes a weighted anti pre-nucleolus of game.

weightedAnti_PreNucl_llp - Computes a weighted anti pre-nucleolus of game.

weightedB0_balancedCollectionQ - Checking a weighted weak Kohlberg's criterion.

weightedBalancedCollectionQ - Checking a weighted Kohlberg's criterion.

weightedCsB0_balancedCollectionQ - Verifies whether the set of induced coalitions is a B0_balanced collection w.r.t. the coalition structure cs.

weightedCsBalancedCollectionQ - Verifies whether the set of induced coalitions is a weighted balanced collection w.r.t. the coalition structure cs.

weightedCsKernel - Computes a weighted kernel element w.r.t. coalition structure cs.

weightedCsKernelQ - Checks whether the imputation x is a weighted kernel element of the TU-game v w.r.t. coalition structure cs.

weightedCsNucl - Computes a weighted nucleolus of game v w.r.t. coalition structure cs.

weightedCsPreKernel - Computes a weighted pre-kernel element w.r.t. a coalition structure cs.

weightedCsPreNucl - Computes a weighted pre-nucleolus of game v w.r.t. coalition structure cs.

weightedGraphGame - Computes from a weighted graph problem a weighted graph TU game.

weightedKernel - Computes a weighted kernel point using optimization toolbox.

weightedKernelQ - Checks if an imputation is a weighted kernel point.

weightedNucl - Computes a weighted nucleolus using optimization toolbox.

weightedNucl_llp - Computes a weighted nucleolus using optimization toolbox.

weightedPreKernel - Computes a weighted prekernel element.

weightedPreNucl - Computes a weighted prenucleolus using optimization toolbox.

weightedPreNucl_llp - Computes a weighted prenucleolus using optimization toolbox.

weighted_cfr_Kernel - Computes a weighted Kernel element with coalition formation restrictions.

weighted_cfr_PreKernel - Computes a weighted Pre-Kernel element with coalition formation restrictions.

weighted_cfr_PreNucl - Computes the weighted pre-nucleolus of game v with coalition formation restrictions.

weighted_cfr_balancedCollectionQ - Verifies whether the set of induced coalitions is a balanced collection with coalition formation restrictions.

weighted_cfr_nucl - Computes the weighted nucleolus of game v with coalition formation restrictions.

weighted_cfr_weak_balancedCollectionQ - Verifies whether the set of induced coalitions is a weakly_balanced collection with coalition formation restriction.

weighted_majority - Creates a weighted majority game.

weighted_truncated - Computes from the threshold th and the weights w_vec a truncated weighted majority game.

winning_coalitions - Determines the whole set of winning coalitions.

winning_players - Computes from a pre-defined set of winning coalitions (e.g. minimal winning coalitions) the set of winning players.

zero_monotonicQ - Checks zero monotonicity.

zero_normalization - Creates a zero normalized game.

Class Objects

TuACore	- subclass object of TuSol (anti-core plot).
TuAPrn	- subclass object of TuSol (anti pre-nucleolus from various solvers).
TuASol	- subclass object of TuGame (game solutions).
TuAVert	- subclass object of TuSol (anti-core vertices).
TuCons	- subclass object of TuSol (consistency).
TuCore	- subclass object of TuSol (core plot).
TuGame	- to perform several computations for retrieving and modifying game data.
TuKcons	- subclass object of TuSol (generalized consistency).
TuKrn	- subclass object of TuSol (kernel solutions from various solvers).
TuMCnets	- subclass object of TUGAME from getting an MC-nets representation game data.
TuNuc	- subclass object of TuSol (nucleolus from various solvers).
TuPrk	- subclass object of TuSol (pre-kernel solutions from various solvers).
TuPrn	- subclass object of TuSol (pre-nucleolus from various solvers).
TuProp	- subclass object of TuGame (game properties).
TuRep	- subclass object of TuSol (prk replication).
TuShRep	- subclass object of TuSol (Shapley value replication).
TuSol	- subclass object of TuGame (game solutions).
TuVal	- subclass object of TuGame (fairness and related values).
TuVert	- subclass object of TuSol (core vertices).

Main Functions: Parallel Computing

p_ADvalue - Computes the Aumann-Dreze value.

p_AP_DummyPlayer_propertyQ - Checks if the solution x satisfies the AP-Dummy player property.

p_AP_DummyPlayers - Returns the player who are AP-Dummy players.

p_AP_NullPlayer_propertyQ - Checks if the solution x satisfies the AP-Null player property.

p_AP_NullPlayers - Returns the players who are AP-Null players.

p_A_DummyPlayer_propertyQ - Checks if the solution x satisfies the A-Dummy player property.

p_A_NullPlayer_propertyQ - Checks if the solution x satisfies the A-Null player property.

p_A_NullPlayers - Returns the players who are A-Null players.

p_AllMarginalContributions - Computes all marginal contributions of a Tu-game.

p_AlmostConcave_gameQ - Returns true whenever the game v is almost concave.

p_AlmostConvex_gameQ - Returns true whenever the game v is almost convex.

p_AntiB0_balancedCollectionQ - Checks the reversal of weighted Kohlberg's criterion.

p_AntiReduced_game_propertyQ - Checks whether an imputation x satisfies the anti-reduced game property.

p_Anti_ChiValue - Computes the anti-chi-value of a TU-game v.

p_Anti_Converse_DGP_Q - Checks whether an imputation x satisfies the anti-converse derived game property.

p_Anti_Derived_game_propertyQ - Checks whether an imputation x satisfies a modified anti-derived game property.

p_Anti_Gap - Computes the anti-gap function from game v.

p_Anti_GenGap - Computes the anti-generalized gap function from game v.

p_Anti_ModPreKernel - Computes from (v,x) a modified pre-kernel element.

p_Anti_ModPrekernelQ - Checks whether the imputation x is a modified anti-pre-kernel element of the TU-game v.

p_Anti_PModPreKernel - Computes from (v,x) a proper modified anti-pre-kernel element.

p_Anti_PModPrekernelQ - Checks whether the imputation x is a proper modified anti-pre-kernel element of the TU-game v.

p_Anti_PreKernel - Computes an anti-pre-kernel element.

p_Anti_PrekernelQ - Checks if an imputation is an anti prekernel point.

p_Anti_PropModPreKernel - Computes from (v,x) a proper modified anti-pre-kernel element.

p_Anti_TauValue - Computes the anti-tau-value of a TU-game v.

p_B0_balancedCollectionQ - Checking weak Kohlberg's criterion.

p_BanzhafColeman - Computes the Banzhaf/Coleman index of a simple game sv using MATLAB's PCT.

p_BanzhafOwenValue - Computes the Banzhaf-Owen value w.r.t. a priori unions cs using Matlab's PCT.

p_BanzhafPenrose - Computes the Banzhaf/Penrose and Banzhaf/Coleman index of a simple game sv using MATLAB's PCT.

p_BestCoalitions - Computes the set of most effective coalitions.

p_COV_propertyQ - Verifies if the payoff x satisfies COV property.

p_CddTotallyBalancedQ - Checks whether the core of all subgames is non-empty (cddmex).

p_ChiValue - Computes the chi-value of a TU-game v.

p_CmpConsistencyQ - Checks whether an imputation x satisfies the complement consistency.

p_CmpRedGame - Computes from (v,x,S) a complement reduced game vS on S at x for game v.

p_CoalitionSolidarity - Determines the coalition solidarity value.

p_ColemanOwenValue - Computes the Coleman-Owen value w.r.t. a priori unions cs using Matlab's PCT. Is not normalized to one.

p_Complement_Reduced_game - Computes from (v,x) all complement reduced games on S at x of game v.

p_Converse_CmpConsistencyQ - Checks whether an imputation x satisfies the converse complement consistency property.

p_Converse_DGP_Q - Checks whether an imputation x satisfies the converse derived game property.

p_Converse_RGP_Q - Checks if an imputation satisfies the CRGP.

p_DM_AntiReduced_game - Computes from (v,x) all anti-reduced games on S at x of game v.

p_DM_Anti_Derived_game - Computes from (v,x) a modified Davis-Maschler anti-reduced game vS on S at x for game v.

p_DM_Derived_game - Computes from (v,x) a modified Davis-Maschler reduced game vS on S at x for game v.

p_DM_Reduced_game - Computes all Davis-Maschler reduced games.

p_DM_TwoReduced_game - Computes from (v,x) all single and two-person reduced games on S at x of game v using MATLAB's PCT.

p_DecomposeGame - Computes the unique decomposition of a TU-game.

p_DeeganPackel - Computes the Deegan-Packel index from the set of minimal winning coalitions.

p_DeeganPackel_SV - Computes the Deegan-Packel index from a simple game to construct the set of minimal winning coalitions.

p_Derived_game_propertyQ - Checks whether an imputation x satisfies a modified derived game property.

p_DualCover - The maximum characteristic values from the primal or dual game.

p_DualEssentialSet - Computes the set of dually essential coalitions.

p_DualFloor - The minimum characteristic values from the primal or dual game.

p_DuallyEssentialSet - Computes the set of dually essential coalitions.

p_DummyPlayer_propertyQ - Verifies if x satisfies the dummy player property.

p_DummyPlayers - Returns the list of dummy players of game v.

p_ECCoverGame - Computes from (v,x) an excess comparability cover of game v.

p_ECFloorGame - Computes from (v,x) an excess comparability floor of game v.

p_ECGValue - Computes the Equal Collective Gains value of a TU-game v.

p_EC_DGP_Q - Checks whether the solution x satisfies excess comparability for each derived game.

p_EC_RGP_Q - Checks whether the solution x satisfies excess comparability for each reduced game.

p_EC_propertyQ - Checks whether the solution x satisfies excess comparability.

p_Gap - Determines the gap function.

p_GenGap - Computes the generalized gap function from game v.

p_GetMarketGame - Determines from a random generated game the corresponding market game.

p_HMS_AntiReduced_game - Computes from (v,x) all Hart/Mas-Colell anti-reduced games on S at x of game v.

p_HMS_Anti_Derived_game - Computes from (v,x,S) a modified Hart-Mas-Colell anti-reduced game vS on S at x for game v.

p_HMS_Derived_game - Computes from (v,x,S) a modified Hart-Mas-Colell reduced game vS on S at x for game v.

p_HMS_ImputSavingReducedGame - Computes from (v,x) all Hart/Mas-Colell ISR games.

p_HMS_Reduced_game - Creates all Hart/Mas-Colell reduced games.

p_HMS_TwoReduced_game - Computes from (v,x) all Hart/Mas-Colell singleton and two-person reduced games on S at x of game v.

p_ISRG_propertyQ - Checks whether an imputation x satisfies the ISR game property.

p_ImputSavingReducedGame - Computes from (v,x) all imputation saving reduced games.

p_InessGame - Computes the inessential game from a payoff vector.

p_Johnston - Computes the Johnston power index from the set of winning coalitions.

p_Kernel - Computes a kernel point using the optimization toolbox.

p_LED_RGPvsDGP - Determines the shift of reduced games of the ECC game or reduced game of the dual cover restricted to N w.r.t. the derived games.

p_LS_Nucl - Computes the least square nucleolus of a game.

p_LS_PreNucl - Computes the least square pre-nucleolus of a game.

p_LedcoconsQ - Checks whether an imputation x satisfies large excess difference converse consistency.

p_Ledcons_propertyQ - Checks whether an imputation x satisfies the ledcons property.

p_MarketGameQ - Checks whether the game v is a market game.

p_MaxConsistencyQ - Checks whether an imputation x satisfies maximal consistency.

p_ModDeeganPackel - Computes the modified Deegan-Packel index from the set of winning coalitions.

p_ModDeeganPackel_SV - Computes the Deegan-Packel index from a simple game to construct the set of minimal winning coalitions.

p_ModHoller - Computes the modified Holler index from the set of winning coalitions.

p_ModPGI - Computes the modified public good index from the set of winning coalitions.

p_ModPGI_SV - Computes the modified public good index from a simple game to determine the set of minimal winning coalitions.

p_ModPreKernel - Computes from (v,x) a modified pre-kernel element.

p_ModPrekernelQ - Checks whether the imputation x is a modified pre-kernel element.

p_MyersonValue - Computes the Myerson value of a Tu game.

p_NetworkBanzhaf - Computes the network Banzhaf power index from the set of winning coalitions of a network E while imposing a threshold of th.

p_NetworkDeeganPackel - Computes the network Deegan-Packel index from the set of winning coalitions of a network E while imposing a threshold of th.

p_NetworkJohnston - Computes the network Johnston power index from the set of winning coalitions of a network E while imposing a threshold of th.

p_NetworkMajorityGame - Computes from a network problem (E,c,th) a network majority TU game (simple game).

p_NetworkModDeeganPackel - Computes the network modified Deegan-Packel index from the set of winning coalitions of a network E while imposing a threshold of th.

p_NetworkModPGI - Computes the network modified public good index from the set of winning coalitions of a network E while imposing a threshold.

p_NetworkPGI - Computes the network public good index from the set of minimal winning coalitions of a network E while imposing a threshold.

p_NetworkShapleyShubik - Computes the network Shapley-Shubik power index from the set of winning coalitions of a network E while imposing a threshold of th.

p_NullPlayer_propertyQ - Verifies if x satisfies the null player property.

p_NullPlayers - Returns the list of null players of game v.

p_OwenValue - Computes the Owen value.

p_PGI - Computes the public good index from the set of minimal winning coalitions.

p_PGI_SV - Computes the public good index from a simple game to determine the set of minimal winning coalitions.

p_PModPreKernel - Computes from (v,x) a proper modified pre-kernel element.

p_PModPrekernelQ - Checks whether the imputation x is a proper modified pre-kernel element.

p_PermutationGame - Computes from an assignment matrix the permutation game.

p_PositionValue - Computes the position value.

p_PreKernel - Computes a pre-kernel element.

p_PrekernelQ - Checks if an imputation is a pre-kernel point.

p_PropModPreKernel - Computes from (v,x) a proper modified pre-kernel element.

p_REAS_propertyQ - Checks if the vector x satisfies the reasonableness on both sides.

p_REC_propertyQ - Checks whether the solution x satisfies reverse excess comparability.

p_Reconfirmation_propertyQ - Checks the RCP.

p_RedGame - Creates a Davis-Maschler reduced game.

p_Reduced_game_propertyQ - Checks the RGP.

p_SD_ShapleyValue - Computes the surplus division Shapley value.

p_SedcoconsQ - Checks whether an imputation x satisfies small excess difference converse consistency.

p_Sedcons_propertyQ - Checks whether an imputation x satisfies the sedcons property.

p_ShapleyValue - Computes the Shapley value (potential).

p_ShapleyValueLB - Computes the Shapley value from the linear basis.

p_ShapleyValueM - Computes the Shapley value while relying on all marginal contributions.

p_SolidarityShapleyValue - Determines the solidarity Shapley value.

p_SolidarityValue - Determines the solidarity value.

p_StrConverse_DGP_Q - Checks whether an imputation x satisfies the strong converse derived game property.

p_StrConverse_RGP_Q - Checks whether an imputation x satisfies the strong CRGP.

p_StrLedcoconsQ - Checks whether an imputation x satisfies satisfies strong large excess difference converse consistency.

p_StrSedcoconsQ - Checks whether an imputation x satisfies satisfies strong small excess difference converse consistency.

p_StrategicEquivalentPrK - Computes the pre-kernel of game v from a strategic equivalent game.

p_SubSets - Creates all subsets of super set.

p_TauValue - Computes the Tau value.

p_UpperPayoff - Computes the utopia and minimum claim vector of game v.

p_UtopiaPayoff - Computes the utopia and minimum claim vector of game v.

p_WSysBestCoalitions - Computes the set of most effective coalitions w.r.t. a weight system.

p_WSys_game_space - Computes a game space w.r.t. a weight system which replicates a payoff as a weighted pre-kernel element.

p_WSys_game_space_red - Computes a game space w.r.t. a weight system which replicates a payoff as a weighted pre-kernel element.

p_WSys_replicate_prk - Replicates a weighted pre-kernel point of a game space w.r.t. a weight system.

p_WeakReduced_game_propertyQ - Checks whether an imputation x satisfies the weak reduced game property (consistency)

p_airport_profit - Computes from a cost and benefit vector the associated surplus game.

p_apu_SolidarityValue - Determines the solidarity value w.r.t. a priori unions.

p_assignment_game - Creates an assignment game.

p_average_concaveQ - Returns true whenever the game v is average-concave.

p_average_convexQ - Checks on average convexity.

p_balancedSetQ - Verifies whether the set of induced coalitions is a balanced collection.

p_banzhaf - Computes the Banzhaf value.

p_basis_coordinates - Determines the basis coordinates of a Tu game.

p_basis_game - Determines bases games.

p_belongToAllSubCores - Checks whether all projections of an imputation x are a member of an associated core of a subgame.

p_bint_AssignmentGame - Creates an assignment game (bintprog).

p_bs_PreKernel - Computes from (v,x) a pre-kernel element using Matlab's PCT and backslash operation.

p_clp_kernel - Computes a kernel point using the CLP solver.

p_clp_weightedKernel - Computes a weighted kernel point using the CLP solver.

p_cls_kernel - Computes a kernel point using the CLS solver.

p_cls_weightedKernel - Computes a weighted kernel point using the CLS solver.

p_coeff_linearbasis - Determines the coefficients (dividends) of a linear basis from a TU game.

p_convex_gameQ - Checks on convexity.

p_coreQ - Checks the non-emptiness of the core.

p_cplex_AssignmentGame - Creates an assignment game using the CPLEX solver.

p_cplex_exact_game - Computes the exact game from v using the CPLEX solver.

p_cplex_kernel - Computes a kernel point using the CPLEX solver.

p_cplex_prekernel - Computes a prekernel point using the CPLEX solver.

p_cplex_weightedKernel - Computes a weighted kernel point using the CPLEX solver.

p_cplex_weightedPreKernel - Computes a weighted prekernel point using the CPLEX solver.

p_cvx_kernel - Computes a kernel point using the CVX solver.

p_cvx_prekernel - Computes a prekernel point using the CVX solver.

p_cvx_weightedKernel - Computes a weighted kernel point using the CVX solver.

p_cvx_weightedPreKernel - Computes a weighted prekernel point using the CVX solver.

p_disagreement - Computes the disagreement vector of game v.

p_equal_treatmentQ - Checks if a vector x satisfies ETP.

p_essentialSet - Computes the set of essential coalitions.

p_exact_game - Computes the exact game from v using Matlab's Optimization toolbox.

p_excess - Computes the excesses.

p_flow_game - Computes from a flow problem a TU flow game using the optimization toolbox.

p_game_basis - Computes a game basis of the n-person TU-game space.

p_game_space - Computes the game space which replicates a payoff as a pre-kernel element.

p_game_space_red - Computes the game space which replicates a payoff as a pre-kernel element.

p_genUnionStable - Creates a union stable system.

p_getMinimalWinning - Computes from a simple game the minimal winning coalitions.

p_getgame - Creates a Tu-game from the unanimity coordinates.

p_glpk_exact_game - Computes the exact game from v using the GLPK solver.

p_glpk_kernel - Computes a kernel point using the GLPK solver.

p_glpk_prekernel - Computes a prekernel point using the GLPK solver.

p_glpk_weightedKernel - Computes a weighted kernel point using the GLPK solver.

p_glpk_weightedPreKernel - Computes a weighted prekernel point using the GLPK solver.

p_grMaxFlowGame - Computes from a flow problem a TU flow game.

p_gurobi_AssignmentGame - Creates an assignment game using the GUROBI solver.

p_gurobi_exact_game - Computes the exact game from v using the GUROBI solver.

p_gurobi_flow_game - Computes from a flow problem a TU flow game (GUROBI).

p_gurobi_kernel - Computes a kernel point using the GUROBI solver.

p_gurobi_prekernel - Computes a prekernel point using the GUROBI solver.

p_gurobi_weightedKernel - Computes a weighted kernel point using the GUROBI solver.

p_gurobi_weightedPreKernel - Computes a weighted prekernel point using the GUROBI solver.

p_harsanyi_dividends - Determines the the unanimity coordinates.

p_holler - Computes the Holler index.

p_homogeneous_representationQ - Checks if the weighted majority game possesses a homogeneous representation.

p_hsl_prekernel - Computes a prekernel point using HSL solvers.

p_hsl_weightedPreKernel - Computes a weighted prekernel point using HSL solvers.

p_ipopt_kernel - Computes a kernel point using the IPOPT solver.

p_ipopt_prekernel - Computes a prekernel point using the IPOPT solver.

p_ipopt_weightedKernel - Computes a weighted kernel point using the IPOPT solver.

p_ipopt_weightedPreKernel - Computes a weighted prekernel point using the IPOPT solver.

p_k_Converse_RGP_Q - Checks if an imputation satisfies the k-CRGP.

p_k_Reconfirmation_propertyQ - Checks the k-RCP.

p_k_Reduced_game_propertyQ - Checks the k-RGP.

p_k_StrConverse_RGP_Q - Checks the strong k-CRGP.

p_k_convexQ - Checks k-convexity of the Tu-game.

p_k_cover - Determines from the Tu-game the corresponding k-game.

p_landlord - Computes a production game arising from l-landlords and t-tenants.

p_lin_prekernel - Computes a prekernel point using optimization toolbox.

p_lin_weightedPreKernel - Computes a weighted prekernel point using optimization toolbox.

p_linear_basis - Determines the linear basis of the n-person TU game space.

p_mcst_game - Computes from a cost matrix the corresponding mcst game.

p_min_aspiration - Computes the minimum aspiration level of players of game v.

p_minimal_representation - Computes from the threshold th and the weights w_vec the minimal representation of an homogeneous weighted majority game.

p_minimal_winning - Computes the minimal winning coalitions.

p_monotone_gameQ - Checks monotonicity of the Tu-game.

p_msk_AssignmentGame - Creates an assignment game using the MOSEK solver.

p_msk_bintAssignmentGame - Computes from an assignment problem the corresponding symmetric assignment game.

p_msk_exact_game - Computes the exact game from v using the MOSEK solver.

p_msk_kernel - Computes a kernel point using the MOSEK solver.

p_msk_prekernel - Computes a prekernel point using the MOSEK solver.

p_msk_weightedKernel - Computes a weighted prekernel point using the MOSEK solver.

p_msk_weightedPreKernel - Computes a weighted kernel point using the MOSEK solver.

p_nullShapley - Determines a basis of the null space for the Shapley-value for n-persons.

p_oases_kernel - Computes a kernel point using the OASES solver.

p_oases_prekernel - Computes a prekernel point using the OASES solver.

p_oases_weightedKernel - Computes a weighted kernel point using the OASES solver.

p_oases_weightedPreKernel - Computes a weighted prekernel point using the OASES solver.

p_ols_prekernel - Computes a prekernel point using optimization toolbox.

p_ols_weightedPreKernel - Computes a weighted prekernel point using optimization toolbox.

p_one_concaveQ - Checks whether the game v is 1-concave.

p_one_convexQ - Checks whether the game v is 1-convex.

p_parity_basis - Computes a basis of the n-person TU game space.

p_parity_coeff - Computes the parity transform of the TU-game v.

p_potential - Determines the potential of a TU game (basis).

p_proper_amount - Computes the largest amount players can contribute to a proper coalition.

p_pure_overhead - Creates the matrix of pure overhead games.

p_qpBB_kernel - Computes a kernel point using the QPBB solver.

p_qpBB_weightedKernel - Computes a weighted kernel point using the QPBB solver.

p_qpc_kernel - Computes a kernel point using the QPC solver.

p_qpc_prekernel - Computes a prekernel point using the QPC solver.

p_qpc_weightedKernel - Computes a weighted prekernel point using the QPC solver.

p_qpc_weightedPreKernel - Computes a weighted kernel point using the QPC solver.

p_qrg_prekernel - Computes a prekernel point using qrginv instead of pinv.

p_qrg_weightedPreKernel - Computes a weighted prekernel point using qrginv instead of pinv.

p_reasonable_outcome - Determines the reasonable outcome.

p_replicate_Shapley - Replicates the Shapley value for a game space.

p_replicate_prk - Replicates a pre-kernel solution as a pre-kernel of a game space.

p_secDiffOperatorQ - Checks whether the second order differences are all positive which indicates convexity.

p_select_starting_pt - Selects a starting point for the pre-kernel computation.

p_semi_convexQ - Checks semi-convexity.

p_smallest_amount - Computes the smallest amount vector of the game.

p_sub_additiveQ - Returns true whenever the game v is sub additive.

p_substitutes - Establishes which pair of players are substitutes.

p_super_additiveQ - Checks the Tu-game on super additivity.

p_superadditive_cover - Computes from game v its superadditive cover.

p_totallyBalancedCoverQ - Checks whether the characteristic function v is equal to a totally balanced cover.

p_totallyBalancedQ	- Checks whether the core of all subgames is non-empty.
p_tricameral_assembly	- Computes from a set of parameters a simple game.
p_unanimity_games	- Computes the unanimity coordinates.
p_union_stableQ	- Checks whether a system is union stable.
p_veto_rich_players	- Returns a list of veto players for the TU-game v.
p_weightedAnti_PreKernel	- Computes a weighted anti-prekernel point.
p_weightedKernel	- Computes a weighted kernel point using the optimization toolbox.
p_weightedPreKernel	- Computes a weighted pre-kernel element.
p_weighted_DualEssentialSet	- Computes the set of weighted dually essential coalitions.
p_weighted_DuallyEssentialSet	- Computes the set of weighted dually essential coalitions.
p_weighted_EssentialSet	- Computes the set of weighted essential coalitions using Matlab's PCT.
p_zero_monotonicQ	- Checks zero monotonicity.

Class Objects

p_TuCons	- subclass object of p_TuSol (consistency).
p_TuKcons	- subclass object of p_TuSol (generalized consistency).
p_TuKrn	- subclass object of p_TuSol (kernel solutions from various solvers).
p_TuPrk	- subclass object of p_TuSol (pre-kernel solutions from various solvers).
p_TuProp	- subclass object of TuGame (game properties).
p_TuRep	- subclass object of p_TuSol (prk replication).
p_TuShRep	- subclass object of p_TuSol (Shapley value replication).

p_TuSol - subclass object of TuGame (game solutions).

p_TuVal - subclass object of TuGame (fairness and related values).

Tools: Sed File

sed_core - Converts cdd file format into Matlab format.