>

# GAME THEORY TEMPLATE

This template solves zero sum (Total Conflict) and non-zero sum (Partial Conflict ) games.

By
**William P. Fox**
**Professor**
**Department of Defense Analysis**
**Naval Postgraduate School**
**Monterey, CA 93943**

```
> Restart :
> with(LinearAlgebra) : with(Optimization) :
>
```

## Total Conflict Games

```
> TotalConflictGame := proc( r, c, A)
        local M, B1, B, X, Y, Cnst1, Cnst, Colin, Rose;
        with(LinearAlgebra) : with(Optimization) :
        #make a local copy of A that we can change
        M := Matrix( A ) : B1 := Transpose(A) : B := -A :
        X := `<,>`(seq(x[i], i=1..r));
        Y := `<,>`(seq(y[i], i=1..c));

        Cnst1 := {seq( (B.Y)[i] ≥ p1 - p2, i=1 .. r) , add(y[i], i=1 .. c) = 1};
        Cnst := { seq( (B1.X)[i] ≥ q1 - q2, i=1 .. c), add(x[i], i=1 .. r) = 1};
        Colin := LPSolve( p1 - p2, Cnst1, assume=nonnegative, maximize);
        Rose := LPSolve(q1 - q2, Cnst, assume=nonnegative, maximize);
        print(Rose, Colin);
    end proc;
```

TotalConflictGame := **proc**(r, c, A) **(1.1)**

```
    local M, B1, B, X, Y, Cnst1, Cnst, Colin, Rose;
    with(LinearAlgebra);
    with(Optimization);
    M := Matrix(A);
    B1 := LinearAlgebra:-Transpose(A);
    B := -A;
    X := < seq(x[i], i=1..r) > ;
    Y := < seq(y[i], i=1..c) > ;
    Cnst1 := {seq(p1 - p2 <= Typesetting:-delayDotProduct(B, Y)[i], i=1..r), add(y[i],
    i=1..c) = 1};
    Cnst := {seq(q1 - q2 <= Typesetting:-delayDotProduct(B1, X)[i], i=1..c), add(x[i],
```

```
    i = 1 ..r) = 1 };
    Colin := Optimization:-LPSolve( p1 − p2, Cnst1, assume = nonnegative, maximize );
    Rose := Optimization:-LPSolve( q1 − q2, Cnst, assume = nonnegative, maximize );
    print( Rose, Colin )
end proc
> # Example 1 Mixed Strategy Game
> A := Matrix( [ [4, -4, 3, 2, -3, 3], [ -1, -1, -2, 0, 0, 4], [ -1, 2, 1, -1, 2, -3 ] ] );
```

$$A := \begin{bmatrix} 4 & -4 & 3 & 2 & -3 & 3 \\ -1 & -1 & -2 & 0 & 0 & 4 \\ -1 & 2 & 1 & -1 & 2 & -3 \end{bmatrix} \tag{1.2}$$

```
> TotalConflictGame( 3, 6, A );
```

$$\left[-0.0714285700799631, \left[q1 = 0., q2 = 0.0714285700799631, x_1 = 0.238095238267338, x_2 = 0.214285714156640, x_3 = 0.547619047576023\right]\right], \left[0.0714285720985307, \left[p1 = 0.0714285720985307, p2 = 0., y_1 = 0., y_2 = 0.357142857050661, y_3 = 0., y_4 = 0.571428571711307, y_5 = 0., y_6 = 0.0714285712380325\right]\right] \tag{1.3}$$

```
> # Example 2 Mixed Strategy Game
> A := Matrix( [ [1, 2, 2 ], [2, 1, 2 ], [2, 2, 0 ] ] );
```

$$A := \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 0 \end{bmatrix} \tag{1.4}$$

```
> ra := 3;
```

$$ra := 3 \tag{1.5}$$

```
> rc := 3;
```

$$rc := 3 \tag{1.6}$$

```
> TotalConflictGame( ra, rc, A );
```

$$\left[1.60000000000000, \left[q1 = 1.60000000000000, q2 = 0., x_1 = 0.400000000000000, x_2 = 0.400000000000000, x_3 = 0.200000000000000\right]\right], \left[-1.59999999710873, \left[p1 = 0., p2 = 1.59999999710873, y_1 = 0.399999997728285, y_2 = 0.400000001858676, y_3 = 0.200000000413039\right]\right] \tag{1.7}$$

```
> # Example 3 Pure Strategy Game
> A := Matrix( [ [12, -1, 1, 0], [5, 1, 7, -20], [3, 2, 4, 3], [ -16, 0, 0, 16 ] ] );
```

$$A := \begin{bmatrix} 12 & -1 & 1 & 0 \\ 5 & 1 & 7 & -20 \\ 3 & 2 & 4 & 3 \\ -16 & 0 & 0 & 16 \end{bmatrix} \tag{1.8}$$

```
> ra := 4 : rc := 4 :
> TotalConflictGame( ra, rc, A );
```

$$\left[2.00000000103260, \left[q1 = 2.00000000103260, q2 = 0., x_1 = 0., x_2 = 0., x_3 = 1., x_4 = 0.\right]\right], [ \tag{1.9}$$

$-1.99999999676835, [p1 = 0., p2 = 1.99999999676835, y_1 = 0., y_2 = 1.00000000090830, y_3 = 0., y_4 = 0.]]$

>

>

## Partial Conflict Games

```
> PartialConflictGame := proc( ar, ac, A, B, ip, iq)
     local X, Y, Cnst, objective;
    with(LinearAlgebra) : with(Optimization) :

    X := `<,>`(seq(x[i], i=1 ..ar));
    Y := `<,>`(seq(y[i], i=1 ..ac));
   Cnst := {seq( (A.Y) [i] ≤ p, i=1 .. ar), seq( (Transpose(X).B) [i] ≤ q, i=1 .. ac),
    add(x[i], i=1 .. ar) = 1, add(y[i], i=1 .. ac) = 1};
    objective := expand(Transpose(X).A.Y + Transpose(X).B.Y-p-q);
    QPSolve(objective, Cnst, assume=nonnegative, maximize, initialpoint = {p = ip, q
   = iq});


 end proc;
```

PartialConflictGame := **proc**(ar, ac, A, B, ip, iq) **(2.1)**

**local** X, Y, Cnst, objective;

with(LinearAlgebra);

with(Optimization);

X := < seq(x[i], i=1 ..ar) > ;

Y := < seq(y[i], i=1 ..ac) > ;

Cnst := {seq(Typesetting:-delayDotProduct(LinearAlgebra:-Transpose(X), B)[i] <= q, i=1 ..ac), seq(Typesetting:-delayDotProduct(A, Y)[i] <= p, i=1 ..ar), add(x[i], i=1 ..ar) = 1, add(y[i], i=1 ..ac) = 1};

objective := expand(Typesetting:-delayDotProduct(Typesetting:-delayDotProduct(LinearAlgebra:-Transpose(X), A), Y) + Typesetting:-delayDotProduct(Typesetting:-delayDotProduct(LinearAlgebra:-Transpose(X), B), Y) − p − q);

Optimization:-QPSolve(objective, Cnst, assume = nonnegative, maximize, initialpoint = {p = ip, q = iq})

**end proc**

```
> # Example 1 Pure Strategy
> A := Matrix([[-1, 0, 0], [2, 1, 0], [0, 1, 2]]);
```

$$A := \begin{bmatrix} -1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 1 & 2 \end{bmatrix} \quad \textbf{(2.2)}$$

```
> B := Matrix([[1, 2, 2], [1, -1, 0], [0, 1, 2]]);
```

$$B := \begin{bmatrix} 1 & 2 & 2 \\ 1 & -1 & 0 \\ 0 & 1 & 2 \end{bmatrix} \tag{2.3}$$

```
> ar := 3 : ac := 3 :
> PartialConflictGame(ar, ac, A, B, 10, 10);
```

$$[0., [p = 2., q = 1., x_1 = 0., x_2 = 1., x_3 = 0., y_1 = 1., y_2 = 0., y_3 = 0.]] \tag{2.4}$$

```
> # Example 2  Equaling Strategy (No pure strategy)
> A := Matrix([[2, 1], [3, 0]]);
```

$$A := \begin{bmatrix} 2 & 1 \\ 3 & 0 \end{bmatrix} \tag{2.5}$$

```
> B := Matrix([[4, 0], [1, 4]]);
```

$$B := \begin{bmatrix} 4 & 0 \\ 1 & 4 \end{bmatrix} \tag{2.6}$$

```
> ar := 2 : ac := 2 :
> PartialConflictGame(ar, ac, A, B, 10, 10);
```

$$[0., [p = 1.50000000000000, q = 2.28571428571429, x_1 = 0.428571428571429, x_2 = 0.571428571428571, y_1 = 0.500000000000000, y_2 = 0.500000000000000]] \tag{2.7}$$

```
>
>
```

# Prudential Strategies and Security Levels

```
> PrudentialStrategies := proc( ar, ac, A, B)
      local X, Y, CnstR, CnstC, Obj1, Obj2, SLR, SLC;
       with(LinearAlgebra) : with(Optimization) :
      X := `<,>`(seq(x[i], i = 1 ..ar));
      Y := `<,>`(seq(y[i], i = 1 ..ac));
      CnstR := {seq((Transpose(X).A)[i] ≥ p1 − p2, i = 1 .. ar), add(x[i], i = 1 .. ar) = 1};
      CnstC := {seq((B.Y)[i] ≥ q1 − q2, i = 1 .. ac), add(y[i], i = 1 .. ac) = 1};
       Obj1 := p1 − p2; Obj2 := q1 − q2;
     SLR := LPSolve(Obj1, CnstR, assume = nonnegative, maximize) :; SLC
      := LPSolve(Obj2, CnstC, assume = nonnegative, maximize);
    print(SLR, SLC);
  end proc;
```

PrudentialStrategies := **proc**(ar, ac, A, B) **(3.1)**

```
    local X, Y, CnstR, CnstC, Obj1, Obj2, SLR, SLC;
    with(LinearAlgebra);
    with(Optimization);
    X := < seq(x[i], i = 1 ..ar) > ;
    Y := < seq(y[i], i = 1 ..ac) > ;
    CnstR := {seq(p1 − p2 <= Typesetting:-delayDotProduct(LinearAlgebra:-
    Transpose(X), A)[i], i = 1 ..ar), add(x[i], i = 1 ..ar) = 1};
    CnstC := {seq(q1 − q2 <= Typesetting:-delayDotProduct(B, Y)[i], i = 1 ..ac), add(y[i
```

```
], i = 1 ..ac) = 1 };
Obj1 := p1 - p2;
Obj2 := q1 - q2;
SLR := Optimization:-LPSolve(Obj1, CnstR, assume = nonnegative, maximize);
SLC := Optimization:-LPSolve(Obj2, CnstC, assume = nonnegative, maximize);
print(SLR, SLC)
end proc
```

```
> # Example 1
> A := Matrix([[3, 8], [4, 6]]);
```

$$A := \begin{bmatrix} 3 & 8 \\ 4 & 6 \end{bmatrix} \quad (3.2)$$

```
> B := Matrix([[7, 5], [5, 6]]);
```

$$B := \begin{bmatrix} 7 & 5 \\ 5 & 6 \end{bmatrix} \quad (3.3)$$

```
> ar := 2 : ac := 2 :
> PrudentialStrategies(ar, ac, A, B);
```

$[4., [p1 = 4., p2 = 0., x_1 = 0., x_2 = 1.]], [5.66666666666667, [q1 = 5.66666666666667, q2 = 0., y_1 = 0.333333333333333, y_2 = 0.666666666666667]]$ **(3.4)**

```
>
>
>
```

# Nash Arbitration Method

```
> NashArbitration := proc( slr, slc, x1, y1, x2, y2)
      local objective, in1;
      with(LinearAlgebra) : with(Optimization) :
     objective := (x - slr) · (y - slc);
```

$$in1 := (y - y1) - \frac{(y1 - y2)}{(x1 - x2)}(x - x1) ;$$

```
     NLPSolve(objective, {in1 = 0, x ≥ slr, y ≥ slc, x ≤ x2, y ≤ y1}, maximize);



  end proc;
```

NashArbitration := **proc**(slr, slc, x1, y1, x2, y2) **(4.1)**

```
    local objective, in1;
    with(LinearAlgebra);
    with(Optimization);
    objective := (x - slr) * (y - slc);
    in1 := y - y1 - (y1 - y2) * (x - x1) / (x1 - x2);
    Optimization:-NLPSolve(objective, {in1 = 0, x <= x2, y <= y1, slr <= x, slc <= y},
    maximize)
```

```
end proc
> p1 := 3 : p2 := 7 : q1 := 8 : q2 := 5 :
> fx := (x - 1 + 0) · (y - (16/7) + 0);
```

$$fx := (x-1)\left(y-\frac{16}{7}\right) \quad \textbf{(4.2)}$$

```
> POl := (y - 4) - (4 - 1)/(2 - 3) · (x - 2) ;
```

$$POl := y - 10 + 3x \quad \textbf{(4.3)}$$

```
> NLPSolve(fx, {POl = 0, x ≥ 1, y ≥ 16/7, x ≥ 2, x ≤ 3, y ≤ 4}, assume = nonnegative,
      maximize);
Warning, no iterations performed as initial point satisfies first-order conditions
```

$$[1.71428571428571441, [x = 2., y = 4.]] \quad \textbf{(4.4)}$$

```
> # Example 1
> NashArbitration(4, 5.6666, 3, 7, 8, 5);
```

$$[0.544522224999998250, [x = 5.16675000000000, y = 6.13330000000000]] \quad \textbf{(4.5)}$$

```
> # Example 2
> NashArbitration(10/3, 6, 4, 8, 10, 5);
```

$$[2.72222222222222010, [x = 5.66666666666667, y = 7.16666666666667]] \quad \textbf{(4.6)}$$

```
>
>
>
```