Mathematica Code For Simplex Method

Zhao Duo (0610133) *
School of Mathematical Sciences, NanKai University,
Tianjin 300071, China.

2008.12.7

Abstract

The software below is based on the two-phase method and the development environment is Mathematica 6.0.3. It is capable of solving all the LP-problems theoretically. Tests from several textbooks justify its correctness, efficiency and robustness. The code below can be divided into three parts. The first part defines all the preliminary functions that will be called in the later part. The second one is the kernel of the application. The function–"cals"–can take record of the entire calculating-process and get the final result. The last part defines the GUI, which makes it more human interactive and convenient. We haven't envelop the first two parts into the last one, thus it is available anywhere in your mathematica dialog boxes after it has been activated. You can use these functions independently.

1 Introductions of the files

The attachment includes three files:
LP.nb (A mathematica file)
LP.tex (A La Tex file)
LP.pdf (Please open with Adobe Reader)
LP sample.JPG(A sample of calculation)

2 The original code

```
(*The first part of the original code, which defines the functions \
that will be called later*) enfont[x_String] :=(*To beautify the
output form*)
  Style[x, FontFamily -> Times, Italic, Bold];
ifallnega[list_] :=
  (For[i = 1; t = True, i <= Length@list, i++,
    If[list[[i]] > 0, t = False; Break[]]]; Return@t);
iftherenon0[list_] :=
  (For[i = 1; t = False, i <= Length@list, i++,
    If[list[[i]] != 0, t = True; Break[]]]; Return@t);
ifthere0[list_] :=
  (For[i = 1; t = False, i <= Length@list, i++,
    If[list[[i]] == 0, t = True; Break[]]]; Return@t);
```

*Corresponding-author. *E-mail addresses*: spacepure@mail.nankai.edu.cn

```
ifnobound[list_, mm_] :=
  (t = False;
   For[j = 1, j <= Length@list, j++,
    If[list[[j]] > 0,
     For[i = 1, i <= Length@mm, i++,
      If[mm[[i, j]] > 0, Break[]]     ];
     If[i == Length@mm + 1, t = True]
     ]
    ]; Return@t);
solu[list1_, list2_, n_] :=
  (*To create an array that represents the solutions*)
  Module[
   {ss = Table[0, {n}]},
   For[i = 1, i <= Length@list1, i++,
    ss[[list1[[i]]]] = list2[[i]]
    ];
   Return@ss
   ];
innum[list_, matrix_, j_] :=
  Module[
   {
    m = Length[list],
    tempmin = -8,
    tempnum = -1
    },
   For[i = 1, i <= m, i++,
    If[matrix[[i, j]] > 0,
     tempnum = i;
     tempmin = list[[i]]/matrix[[i, j]];
     Break[],
     Continue[]]
    ];
   If[i > m, Print["No positive numbers in the column"],
    For[i = tempnum, i <= m, i++,
     If[matrix[[i, j]] > 0,
      If[list[[i]]/matrix[[i, j]] <= tempmin,
       tempnum = i;
       tempmin = list[[i]]/matrix[[i, j]]],
      Continue[]
      ]
     ]];
   Return[tempnum  ]
   ];
r1matrix[bb_, mm_, {i_, j_}] :=
  Module[
   {
    m = Dimensions[mm][[1]],
    n = Dimensions[mm][[2]]
    },
   nextmatrix = Table[0, {m}, {n}];
   nextbb = Table[0, {m}];
   If[mm[[i, j]] != 0,
```

```
      nextmatrix[[i]] = mm[[i]]/mm[[i, j]];
      nextbb[[i]] = bb[[i]]/mm[[i, j]],
      Print["Not a non-zero number"]];
    For[k = 1, k <= m, k++,
     If[k == i, Continue[],
       nextmatrix[[k]] = mm[[k]] - mm[[k, j]]*nextmatrix[[i]]];
      nextbb[[k]] = bb[[k]] - mm[[k, j]]*nextbb[[i]]
      ];
    Return[{nextbb, nextmatrix}]
    ];
findmaxnum[list_, num_] :=
  Module[
   {
    maxtemp = num[[1]],
    end = Length[num]
    },
   For[i = 2, i <= end, i++,
    If[list[[num[[i]]]] > list[[maxtemp]],
     maxtemp = num[[i]]
      ]
    ];
   Return[maxtemp]
   ];
transback[list_, memo_] :=
  (*Tranform the the solutions to the original form through elimating \
the non-original variables *)
  Module[
   {s = Table[0, {Length@memo}]},
   For[i = 1, i <= Length@memo, i++,
    If[memo[[i]] == 0, s[[i]] = list[[i]],
     If[memo[[i]] == -1, s[[i]] = -list[[i]],
      s[[i]] = list[[i]] - list[[memo[[i]]  ]]
      ]]
    ]; Return@s
   ];
stringstatus[sta_] :=
  Piecewise[
   {
    {enfont@"No feasible solutions", sta == -1},
    {enfont@"Still calculating", sta == 0},
    {enfont@"An unique optimal solution", sta == 1},
    {enfont@"Multiple optimal solutions", sta == 2},
    {enfont@"Unbounded", sta == 3}
    }
   ];
outsigns[i_] :=
  Piecewise[{{"\[LessEqual]", i == -1}, {"=",
      i == 0}, {"\[GreaterEqual]", i == 1}}];
xoutform[
   list_] := (Style[Subscript[x, #], Bold, Italic,
       FontFamily -> Times] &
     /@ list);
```

```
(*The second part starts here, where the key functions are
achieved*)


cals[c00i_, bases1i_, b1i_, opermatrixi_, artifvi_] :=
 Module[
  {
   m = Dimensions[opermatrixi][[1]],
   n = Dimensions[opermatrixi][[2]],
   c00 = c00i,
   bases1 = bases1i,
   b1 = b1i,
   opermatrix = opermatrixi,
   c1 = {},
   s = 0,
   swap1 = {0, 0},
   checks1 = {},
   status1 = 0,
   artifv = artifvi,
   variables = {},
   datas1 = {},
   swaplist1 = {{0, 0}},
   step = 0,
   solutions = {},
   tempsolu = 0,
   memoremain = {},
   tempremain = {},
   nonbases1 = {}
   },
  (*To initialise the first section*)
  step++;
  variables = Range[n];
  nonbases1 = Complement[Table[i, {i, 1, n}], bases1];
  swaplist1 = {{0, 0}};
  If[(s = Length@artifv) != 0,
   c00 = Table[0, {n}];
   For[i = 1, i <= s, i++,
    c00[[artifv[[i]]]] = -1]];
  c1 = c00[[#]] & /@ bases1;
  checks1 =
   Table[
    c00[[j]] - Sum[c1[[i]]*opermatrix[[i, j]], {i, 1, m}], {j, 1,
     n}];
  tempsolu = solu[bases1, b1, Dimensions[opermatrix][[2]]];
  AppendTo[solutions, {tempsolu.c00, tempsolu}];
  If[ifallnega[checks1],
   If[iftherenon0[Intersection[bases1, artifv]],
    status1 = -1,
    If[ifthere0[checks1[[#]] & /@ nonbases1],
     status1 = 2, status1 = 1]],
   If[ifnobound[checks1, opermatrix],
    status1 = 3, status1 = 0]
```

```
  ];
AppendTo[datas1, Grid[ArrayFlatten@
    {{{{"c", SpanFromLeft, SpanFromLeft}}, {c00}},
     {{{SpanFromAbove, SpanFromBoth,
        SpanFromBoth}}, {Style[Subscript[x, #], Bold, Italic,
          FontFamily -> Times] & /@ Range[n]}},
     {Transpose@{c1,
        Style[Subscript[x, #], Italic, FontFamily -> Times] & /@
         bases1, b1}, opermatrix},
     {{{"\[Sigma]", SpanFromLeft, SpanFromLeft}}, {checks1}}},
   Background -> {None, None, {0, 0} -> Green}, Frame -> All,
   FrameStyle -> Gray,
   Dividers -> {
     Rule[#, {Thickness[2.2], Blue}] & /@ {1, 4, -1},
     Rule[#, {Thickness[2.2], Blue}] & /@ {1, 2, 3, -1, -2}
     },
   ItemSize -> {3, 2}   ]     ];
(*The first Loop is starting,
meanwhile the corresponding tableau will be created*)
While[
 status1 == 0,
 swap1[[2]] = findmaxnum[checks1, nonbases1];
 swap1[[1]] = innum[b1, opermatrix, swap1[[2]]];
 datas1[[step, 2, 2, 3, 1]] = swap1 + {2, 3};
 step++;
 bases1[[swap1[[1]]]] = swap1[[2]];
 nonbases1 = Complement[Table[i, {i, 1, n}], bases1];
 c1 = c00[[#]] & /@ bases1;
 {b1, opermatrix} = r1matrix[b1, opermatrix, swap1];
 checks1 =
  Table[
   c00[[j]] - Sum[c1[[i]]*opermatrix[[i, j]], {i, 1, m}], {j, 1,
    n}];
 tempsolu = solu[bases1, b1, Dimensions[opermatrix][[2]]];
 AppendTo[solutions, {tempsolu.c00, tempsolu}];
 If[ifallnega[checks1],
  If[iftherenon0[Intersection[bases1, artifv]], status1 = -1,
   If[ifthere0[checks1[[#]] & /@ nonbases1], status1 = 2,
    status1 = 1]],
  If[ifnobound[checks1, opermatrix], status1 = 3, status1 = 0]
  ];
 AppendTo[datas1, Grid[ArrayFlatten@
     {{{{"c", SpanFromLeft, SpanFromLeft}}, {c00}},
      {{{SpanFromAbove, SpanFromBoth,
         SpanFromBoth}}, {Style[Subscript[x, #], Bold, Italic,
           FontFamily -> Times] & /@ Range[n]}},
      {Transpose@{c1,
         Style[Subscript[x, #], Italic, FontFamily -> Times] & /@
          bases1, b1}, opermatrix},
      {{{"\[Sigma]", SpanFromLeft, SpanFromLeft}}, {checks1}}},
    Background -> {None, None, {0, 0} -> Green}, Frame -> All,
    FrameStyle -> Gray,
```

```
    Dividers -> {
      Rule[#, {Thickness[2.2], Blue}] & /@ {1, 4, -1},
      Rule[#, {Thickness[2.2], Blue}] & /@ {1, 2, 3, -1, -2}
      },
    ItemSize -> {3, 2}   ]      ]
  ];
(*The second part will begin, if it exists*)
If[s != 0 && status1 != -1,
 step++;
 tempremain = Range[n];
 variables = Complement[variables, artifv];
 For[i = 1, i <= Length@artifv, i++,
  tempremain[[artifv[[i]]]] = 0   ];
 memoremain = Table[0, {n}];
 For[i = 1; j = 1, i <= n, i++,
  If[tempremain[[i]] != 0, memoremain[[i]] = j++]  ];
 n -= s;
 c00 = Delete[c00i, Transpose@{artifv}];
 opermatrix = Delete[#, Transpose@{artifv}] & /@ opermatrix;
 nonbases1 = Complement[nonbases1, artifv];
 c1 =
  Table[
   c00[[      memoremain[[  bases1[[i]]      ]]     ]], {i, 1,
    Length@bases1}];
 checks1 =
  Table[
   c00[[j]] - Sum[c1[[i]]*opermatrix[[i, j]], {i, 1, m}], {j, 1,
    n}];
 tempsolu = solu[bases1, b1, Dimensions[opermatrixi][[2]]];
 AppendTo[solutions, {tempsolu.c00i, tempsolu}];

 If[ifallnega[checks1],
  If[iftherenon0[Intersection[bases1, artifv]], status1 = -1,
   If[
    ifthere0[
     Table[
      checks1[[memoremain[[nonbases1[[i]]]]]], {i, 1,
       Length@nonbases1}]], status1 = 2, status1 = 1]],
  If[ifnobound[checks1, opermatrix], status1 = 3, status1 = 0]
  ];
 AppendTo[datas1, Grid[ArrayFlatten@
    {{{{"c", SpanFromLeft, SpanFromLeft}}, {c00}},
     {{{SpanFromAbove, SpanFromBoth,
        SpanFromBoth}}, {Style[Subscript[x, #], Bold, Italic,
          FontFamily -> Times] & /@ variables}},
     {Transpose@{c1,
        Style[Subscript[x, #], Italic, FontFamily -> Times] & /@
         bases1, b1}, opermatrix},
     {{{"\[Sigma]", SpanFromLeft, SpanFromLeft}}, {checks1}}},
   Background -> {None, None, {0, 0} -> Green}, Frame -> All,
   FrameStyle -> Gray,
   Dividers -> {
```

```
    Rule[#, {Thickness[2.2], Blue}] & /@ {1, 4, -1},
    Rule[#, {Thickness[2.2], Blue}] & /@ {1, 2, 3, -1, -2}
    },
  ItemSize -> {3, 2}   ]      ];
(*Having initialized, it will enter the second loop*)
While[
 status1 == 0,
 swap1[[2]] = findmaxnum[checks1,
   Table[
    memoremain[[nonbases1[[i]]]], {i, 1, Length@nonbases1}]];
 swap1[[1]] = innum[b1, opermatrix, swap1[[2]]];
 datas1[[step, 2, 2, 3, 1]] = swap1 + {2, 3};
 step++;
 bases1[[swap1[[1]]]] = variables[[swap1[[2]]  ]];
 nonbases1 = Complement[variables, bases1];
 c1 =
  Table[
   c00[[      memoremain[[  bases1[[i]]     ]]     ]], {i, 1,
    Length@bases1}];
 {b1, opermatrix} = r1matrix[b1, opermatrix, swap1];
 checks1 =
  Table[
   c00[[j]] - Sum[c1[[i]]*opermatrix[[i, j]], {i, 1, m}], {j, 1,
    n}];
 tempsolu = solu[bases1, b1, Dimensions[opermatrixi][[2]]];
 AppendTo[solutions, {tempsolu.c00i, tempsolu}];
 If[ifallnega[checks1],
  If[iftherenon0[Intersection[bases1, artifv]], status1 = -1,
   If[
    ifthere0[
     Table[
      checks1[[memoremain[[nonbases1[[i]]]]]], {i, 1,
       Length@nonbases1}]], status1 = 2, status1 = 1]],
  If[ifnobound[checks1, opermatrix], status1 = 3, status1 = 0]
  ];
 AppendTo[datas1, Grid[ArrayFlatten@
    {{{{"c", SpanFromLeft, SpanFromLeft}}, {c00}},
     {{{SpanFromAbove, SpanFromBoth,
        SpanFromBoth}}, {Style[Subscript[x, #], Bold, Italic,
          FontFamily -> Times] & /@ variables}},
     {Transpose@{c1,
        Style[Subscript[x, #], Italic, FontFamily -> Times] & /@
         bases1, b1}, opermatrix},
     {{{"\[Sigma]", SpanFromLeft, SpanFromLeft}}, {checks1}}},
   Background -> {None, None, {0, 0} -> Green}, Frame -> All,
   FrameStyle -> Gray,
   Dividers -> {
     Rule[#, {Thickness[2.2], Blue}] & /@ {1, 4, -1},
     Rule[#, {Thickness[2.2], Blue}] & /@ {1, 2, 3, -1, -2}
     },
   ItemSize -> {3, 2}   ]      ]
 ]
```

```
      ];
      Return[{status1, solutions[[step]], solutions, datas1}]
    ];
(*Here is the third part, which defines all the graphic user \
interfaces and makes the application more human interactive*)
DynamicModule[
 {
  m = 0,
  n = 0,
  tempm = 3,
  tempn = 3,
  maxQ = 1,
  maxQ1 = {{1}},
  head1 = {{}},
  object = {},
  object1 = {{}},
  v1 = {{}},
  signsofv = {},
  signsofv1 = {{}},
  num1 = {{}},
  inputmatrix = {{}},
  inputmatrix1 = {{}},
  signsofeq0 = {},
  signsofeq1 = {{}},
  signsofeq = {},
  b0 = {},
  b1 = {{}},
  b = {},
  calmatrix = {{0}},
  c = {},
  nc = 0,
  ncend = 0,
  memo = {},
  bases = {},
  nonbases = {},
  swapnum = {1, 1},
  checknum = {},
  artifv = {},
  artifvend = 0,
  insertrow1 = {{}},
  deleterow1 = {{}},
  insertcol1 = {{}},
  deletecol2 = {{}},
  addnewrow1 = {{}},
  addnewcol1 = {{}},
  start1 = True,
  start2 = False,
  start3 = False,
  opermatrix = {{}}  ,
  operobject = {},
  calresults = {},
  solustatus = 0,
```

```
  finalvalue = 0,
  finalsolus = {},
  originalsolus = {},
  originalvalue = 0,
  initiQ = False
  },
 Column@{
   Grid@
    {
     {
      Text["The number of equations:"],
      InputField[Dynamic[tempm], Number, ImageSize -> {35, 20},
       Enabled -> Dynamic@start1],
      Button["+", If[(++tempm) < 1, tempm = 1],
       Enabled -> Dynamic@start1],
      Button["-", If[(--tempm) < 1, tempm = 1],
       Enabled -> Dynamic@start1],
      Dynamic@If[IntegerQ[tempm], "OK", tempm = IntegerPart[tempm]],
      Dynamic@If[tempm < 1, tempm = 1, "OK"]
      },
     {
      Text["The number of variables:"],
      InputField[Dynamic[tempn], Number, ImageSize -> {35, 20},
       Enabled -> Dynamic@start1],
      Button["+", If[(++tempn) < 1, tempn = 1],
       Enabled -> Dynamic@start1],
      Button["-", If[(--tempn) < 1, tempn = 1],
       Enabled -> Dynamic@start1],
      Dynamic@If[IntegerQ[tempn], "OK", tempn = IntegerPart[tempn]],
      Dynamic@If[tempn < 1, tempn = 1, "OK"]
      },
     {
      Button[
       "Create Input Tableau",
       If[tempm < 1, m = tempm = 1, m = tempm];
       If[tempn < 1, n = tempn = 1, n = tempn];
       maxQ = 1;
       maxQ1 = {{PopupMenu[
            Dynamic[maxQ], {0 -> "min", 1 -> "max"}]}};
       head1 = {{Style["No.", Bold, Italic, FontColor -> Green]}};
       object = Table[0, {n}];
       object1 = {InputField[Dynamic[object[[#]]], Number,
             ImageSize -> {50, 20}] & /@ Range[n]};
       v1 = {Style[Subscript[x, #], Bold, Italic,
             FontFamily -> Times] &
          /@ Range[n]};
       signsofv = Table[1, {n}];
       signsofv1 = {PopupMenu[Dynamic[signsofv[[#]]],
             {-1 -> "\[LessEqual]0", 1 -> "\[GreaterEqual]0" ,
              2 -> "?"}] & /@ Range[n]};
       num1 =
        Table[{Style["(" <> ToString[i] <> ")", Bold,
```

```
    FontFamily -> Times]}, {i, 1, m}];
  inputmatrix = Table[0, {m}, {n}];
  inputmatrix1 = Array[

    InputField[Dynamic@inputmatrix[[#1, #2]], Number,
      Appearance -> Frameless,
      ImageSize -> {25, 20}] &,
    {m, n}];
  signsofeq0 = Table[0, {m}];
  signsofeq1 = Array[
    {PopupMenu[Dynamic@signsofeq0[[#]],
       {-1 -> "\[LessEqual]", 0 -> "=",
        1 -> "\[GreaterEqual]" }]} &, m];
  signsofeq = Table[0, {m}];
  b0 = Table[0, {m}];
  b1 = Array[{InputField[Dynamic@b0[[#]],
       Number, ImageSize -> {50, 20}]} &, m];
  b = Table[0, {m}];
  bases = Table[0, {m}];
  start1 = False;
  start2 = True;
  start3 = False
  ,
  Enabled -> Dynamic@start1
  ],
 Button["Reset",
  start1 = True;
  start2 = False;
  start3 = False;
  m = 0;
  n = 0;
  tempm = 3;
  tempn = 3;
  maxQ = 1;
  maxQ1 = {{1}};
  head1 = {{}};
  object = {};
  object1 = {{}};
  v1 = {{}};
  signsofv = {};
  signsofv1 = {{}};
  num1 = {{}};
  inputmatrix = {{}};
  inputmatrix1 = {{}};
  signsofeq0 = {};
  signsofeq1 = {{}};
  signsofeq = {};
  b0 = {};
  b1 = {{}};
  b = {};
  calmatrix = {{0}};
  c = {};
```

```
      nc = 0;
      ncend = 0;
      memo = {};
      bases = {};
      nonbases = {};
      swapnum = {1; 1};
      checknum = {};
      artifv = {};
      artifvend = 0;
      insertrow1 = {{}};
      deleterow1 = {{}};
      insertcol1 = {{}};
      deletecol2 = {{}};
      addnewrow1 = {{}};
      addnewcol1 = {{}};
      opermatrix = {{}} ;
      operobject = {};
      calresults = {};
      solustatus = 0;
      finalvalue = 0;
      finalsolus = {};
      originalsolus = {};
      originalvalue = 0;
      initiQ = False]
    }
   },
Dynamic@If[m > 0 && n > 0,
   Grid[ArrayFlatten@
     {
      {maxQ1, object1, "#", "#"},
      {"#", v1, "#", "#"},
      {head1, signsofv1, "#", "#"},
      {num1, inputmatrix1, signsofeq1, b1}
      },
    Frame -> All],
   "Not initialized"],
Button
 [
 "To Get Initial matrix",
 nc = n;
 ncend = n;
 signsofeq = signsofeq0;
 For[j = 1, j <= n, j++, If[signsofv[[j]] == 2, ncend++]];
 For[i = 1, i <= m, i++,
   If[signsofeq[[i]]*If[b0[[i]] >= 0, 1, -1] == 1,
    ncend += 2; artifvend++,
    If[signsofeq[[i]]*If[b0[[i]] >= 0, 1, -1] == 0,
     ncend++; artifvend++,
     ncend++
     ]
    ]];
 memo = Table[0, {n}];
```

```
c = Table[0, {ncend}];
artifv = {};
If[maxQ == 1,
 For[j = 1, j <= n, j++,
  c[[j]] = object[[j]]],
 For[j = 1, j <= n, j++,
  c[[j]] = -object[[j]]]
 ];
calmatrix = Table[0, {m}, {ncend}];
For[j = 1, j <= n, j++,
 If[
  signsofv[[j]] == 2,
  memo[[j]] = (++nc);
  For[i = 1, i <= m, i++,
   calmatrix[[i, j]] = inputmatrix[[i, j]];
   calmatrix[[i, nc]] = -inputmatrix[[i, j]]
   ],
  memo[[j]] = 0;
  If[signsofv[[j]] == -1,
   For[
    i = 1, i <= m, i++,
    calmatrix[[i, j]] = -inputmatrix[[i, j]]
    ],
   For[i = 1, i <= m, i++,
    calmatrix[[i, j]] = inputmatrix[[i, j]]
    ]
   ]
  ]
 ]; (* While ends *)
For[i = 1, i <= m, i++,
 If[b0[[i]] < 0,
  b[[i]] = -b0[[i]];
  calmatrix[[i]] = -calmatrix[[i]];
  signsofeq[[i]] = -signsofeq[[i]],
  b[[i]] = b0[[i]]]
 ];
For[i = 1, i <= m, i++,
 If[signsofeq[[i]] == -1,
  calmatrix[[i, ++nc]] = 1;
  bases[[i]] = nc,
  If[signsofeq[[i]] == 0,
   calmatrix[[i, ++nc]] = 1;
   bases[[i]] = nc;
   AppendTo[artifv, nc],
   calmatrix[[i, ++nc]] = -1;
   calmatrix[[i, ++nc]] = 1 ;
   bases[[i]] = nc;
   AppendTo[artifv, nc]
   ]
  ]
 ];
initiQ = True;
```

```
    start3 = True;
    nonbases =
     Complement[Table[i, {i, 1, Dimensions[calmatrix][[2]]}], bases],
    Enabled -> Dynamic@start2
    ], (*Button ends*)
   Style["Standardized:", FontFamily -> Times, Blue, FontSize -> 14,
    Italic],
   Dynamic[Row@{
      If[maxQ == 1,
       "max" {c}.xoutform@Range@ncend,
       "min" {-c}.xoutform@Range@ncend]
      }] ,
   Dynamic@If[initiQ, Grid[Transpose@ArrayFlatten@{
         {{calmatrix.xoutform@Range@ncend}},
         {       {outsigns /@ signsofeq}      },
         {{b}}
         }, Alignment -> Left, ItemSize -> {Automatic, 2}],
     "Not initialized"],
   Dynamic@If[m > 0 && n > 0, Grid[{
      {enfont@"Original variables",
       Item[xoutform@Range[n], ItemSize -> 10]},
      {enfont@"Slack & surplus variables",
       xoutform@Complement[Range[n + 1, ncend], artifv]},
      {enfont@"Artifical variables", xoutform@artifv},
      {enfont@"Basic variables", xoutform@bases},
      {enfont@"Non-basic variables", xoutform@nonbases}
      }, Frame -> All, ItemSize -> {12, 2}]],
   Button["To calculate",
    calresults = cals[c, bases, b, calmatrix, artifv];
    solustatus = calresults[[1]];
    finalvalue = calresults[[2, 1]];
    finalsolus = calresults[[2, 2]];
    originalsolus = transback[finalsolus, memo];
    originalvalue = If[maxQ == 1, finalvalue, -finalvalue];
    Print@Column@{
       Grid[{{enfont["Status"],
          stringstatus[solustatus]}, {enfont[
           "Current optimal value"],
          originalvalue}, {enfont["Current optimal solution"],
          originalsolus}},
        ItemSize -> {12, 2.6}, Frame -> All],
       Column[calresults[[3]], Spacings -> 2],
       Column[calresults[[4]], Spacings -> 2]},
    Enabled -> Dynamic@start3
    ]
   }
 ](*The code ends. All the codes merged into only one cell, Press the \
Shift+Enter and enjoy its convenience and efficency*)
```