

# 22098 Programação Matemática

---

## Texto de apoio 1 – solvers de programação linear

### Índice

Programação Linear – o solver do Excel.....	2
Ativação do solver .....	2
Usando o solver .....	4
Programação inteira e não-linear com o solver do Excel.....	8
Programação linear, inteira e mista – o lpSolve .....	10
lpSolve em ambiente R.....	10
lpSolve com ficheiro de input.....	14

## Programação Linear – o solver do Excel

Na programação linear, a tabela simplex é uma forma eficaz de organizar os cálculos e resolver problemas. Saber executar o algoritmo por essa tabela permite compreender melhor o método simplex, a sua filosofia e interpretação dos resultados. No entanto, quando o problema passa das 3 ou 4 variáveis não-básicas e outras tantas restrições, o cálculo manual torna-se muito demorado e a probabilidade de errar nas contas é alta. Além disso, quando o problema não tem a origem como vértice inicial (solução básica  $\vec{x} = (0,0, \dots)$  não-admissível), há que proceder a uma série de transformações e outros procedimentos preliminares, alguns deles demorados, antes de arrancar com o algoritmo. Juntando isto ao facto de que problemas reais podem ter milhares de variáveis em jogo, rapidamente percebemos que é necessário automatizar cálculos.

Assim, ao longo dos anos foram desenvolvidos diversos métodos de cálculo automático, nos quais se inclui o solver do Excel. A vantagem do solver é que é uma ferramenta de uso intuitivo e direto. Existem solvers mais rápidos e precisos, como o lpSolve, Gurobi ou IBM CPLEX (versão limitada grátis para uso académico), que são desenhados para atacar problemas gigantes, com milhares de variáveis e restrições. A interface destes solvers é mais complicada do que o Excel, mas, se o problema tiver mais de, digamos, 10 variáveis e restrições, acabam por se tornar mais rápidos de programar que o Excel. Começemos, no entanto, pelo solver do Excel, uma vez que este não requer instalar nenhum novo software.

O solver Excel é um *add-on* do Excel que permite resolver problemas de programação linear, inteira e não-linear. No caso da programação linear, o solver devolve soluções exatas, i.e. ótimos exatos. Para a programação inteira e não-linear, isso já não é garantido.

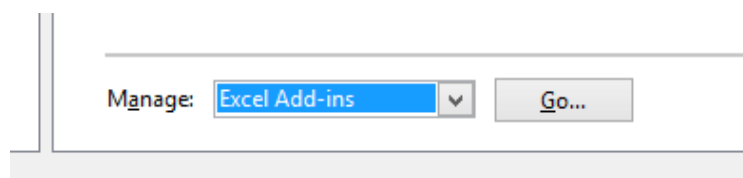
Vejamos então alguns exemplos da utilização deste solver.

### Ativação do solver

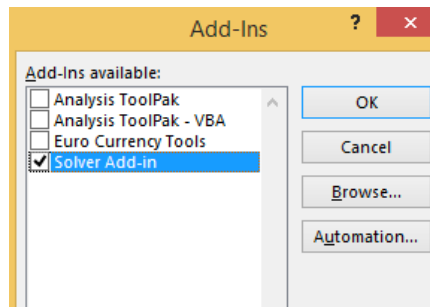
Antes de mais é necessário ativar o solver. Isso faz-se da seguinte forma:

#### Windows:

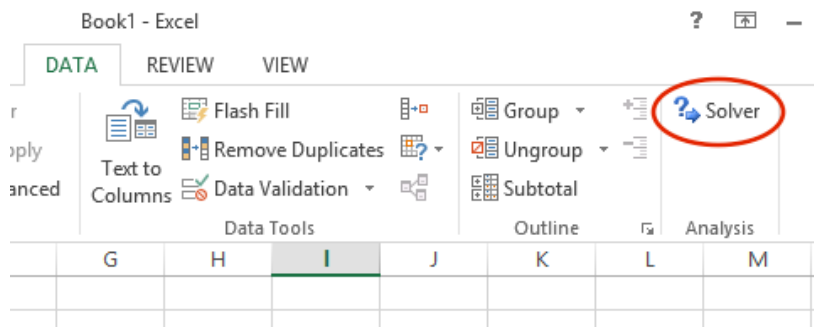
1. Arrancar o Excel (2010+).
2. **File** → **Options** → **Add-Ins** → **Manage Excel Add-Ins** → Selecionar na caixa “**Excel Add-Ins**”




→ clicar **Go** → Checkar “**Solver Add-In**” → clicar **OK**.



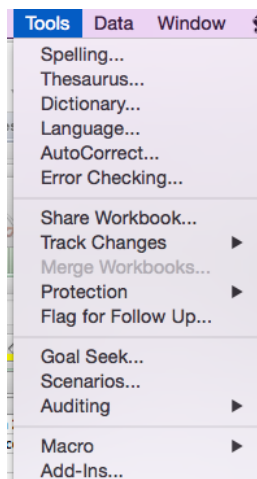
A partir de agora aparecerá um menu novo **Data → Analysis → Solver**:



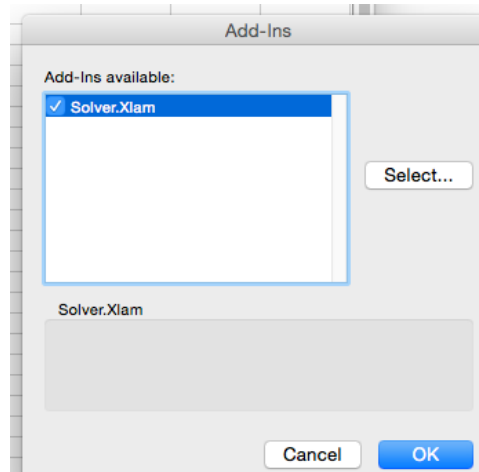
Nota: no Excel 2007 o solver pode ser ativado no botão Microsoft Office: . Para mais informações podem procurar no Google por “Activate solver excel 2007”. Serão redirecionados para o help da Microsoft.

### Mac OS:

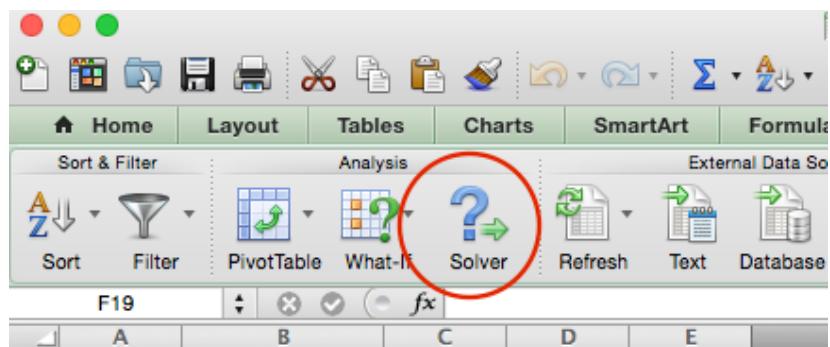
1. Arrancar o Excel.
2. **Tools → Add-Ins**



→ Checkar “solver.xmls” → OK



3. Aparecerá agora no menu **Data** o grupo **Solver**:



### Usando o solver

Para usar o solver basta inserir informação em algumas células da folha de cálculo e clicar no menu *solver*. Há várias maneiras de fazer isso. Aqui fica uma delas.

Suponhamos que deseja resolver o seguinte problema linear:

$$\text{Max } Z = 2x_1 - 3x_2 + 4x_3 - 5x_4$$

$$\text{s. a. } x_1 - 4x_2 \leq 10$$

$$x_2 + 7x_3 \leq 11$$

$$x_3 + 5x_4 \geq 12$$

$$x_1, x_2, x_3, x_4 \geq 0$$

1. Em primeiro lugar, quando se recorre ao solver do Excel, é boa prática escrever numa célula aquilo a que se refere a coluna adjacente. Assim, uma possível estrutura para a entrada de dados seria p.ex.

	A	B	C	D
	Programação Matemática 22098			
	Algoritmo Simplex com Excel			
	Problema do texto de apoio 1			
	x1	0		
	x2	0		
	x3	0		
	x4	0		
	Z	0		
0	restrição 1	0	10	
1	restrição 2	0	11	
2	restrição 3	0	12	

2. Nas colunas das variáveis não-básicas damos um ponto inicial ao algoritmo. Quando usamos a tabela simplex, este é sempre (0,0,...), mas no solver se quisermos podemos escolher outro ponto. A escolha do ponto inicial é mais relevante para problemas não-lineares ou inteiros. Para problemas lineares só faz diferença se o problema for muito grande.

3. Nas colunas de Z e restrições vamos inserir *fórmulas*. Em Z colocamos a função objetivo e nas restrições as expressões destas:

	A	B	C	D
1	Programação Matemática 22098			
2	Algoritmo Simplex com Excel			
3	Problema do texto de apoio 1			
4				
5	x1	0		
6	x2	0		
7	x3	0		
8	x4	0		
9	Z	=2*B5-3*B6+4*B7-5*B8		
10	restrição 1	0		10
11	restrição 2	0		11
12	restrição 3	0		12

	A	B	C	D
1	Programação Matemática 22098			
2	Algoritmo Simplex com Excel			
3	Problema do texto de apoio 1			
4				
5	x1	0		
6	x2	0		
7	x3	0		
8	x4	0		
9	Z	=B5-4*B6		
10	restrição 1	=B5-4*B6		10
11	restrição 2	0		11
12	restrição 3	0		12

	A	B	C	D
1	Programação Matemática 22098			
2	Algoritmo Simplex com Excel			
3	Problema do texto de apoio 1			
4				
5	x1	0		
6	x2	0		
7	x3	0		
8	x4	0		
9	Z	=B6+7*B7		
10	restrição 1	0		10
11	restrição 2	=B6+7*B7		11
12	restrição 3	0		12

	A	B	C	D
1	Programação Matemática 22098			
2	Algoritmo Simplex com Excel			
3	Problema do texto de apoio 1			
4				
5	x1	0		
6	x2	0		
7	x3	0		
8	x4	0		
9	Z	=B7+5*B8		
10	restrição 1	0		10
11	restrição 2	0		11
12	restrição 3	=B7+5*B8		12

É necessário estarem fórmulas tanto na função objetivo como nas restrições, dado que é dessas fórmulas que o solver deduz que problema se quer resolver.

Nas colunas ao lado das restrições pode-se colocar os valores extremos que estas podem tomar. Como veremos, não é obrigatório fazer isto, mas é recomendável porque ajuda a organizar a folha de cálculo.

4. Agora chama-se o solver. Abre-se uma janela e tem-se:

**Solver Parameters**

Set Objective:

To: ☒ Max ☐ Min ☐ Value Of:

By Changing Variable Cells:

Subject to the Constraints:

- 
- 
- 


☒ Make Unconstrained Variables Non-Negative

Select a Solving Method:

**Solving Method**  
Select the GRG Nonlinear engine for Solver Problems that are smooth nonlinear. Select the LP Simplex engine for linear Solver Problems, and select the Evolutionary engine for Solver problems that are non-smooth.

Buttons: Add, Change, Delete, Reset All, Load/Save, Close, Solve

Os parâmetros preenchem-se da seguinte forma: (o quadro poderá variar conforme a língua e versão do Excel)

Em **“Set objective:”** colocamos a célula onde programamos a função objetivo. Não é preciso escrever essa célula por extenso; basta clicar no botão à direita do campo (  ) e depois clicar na célula correspondente.

Em **“To:”** indicamos o teor da otimização: maximizar ou minimizar.

Em **“By Changing Variable cells:”** coloca-se as células onde colocámos o ponto inicial de procura.

Em **“Subject to the Constraints:”**, que da primeira vez aparece em branco, clicamos em **“Add”** e aparece-nos uma janela para indicar as restrições:

**Add Constraint**

Cell Reference:

Constraint:

Operator:

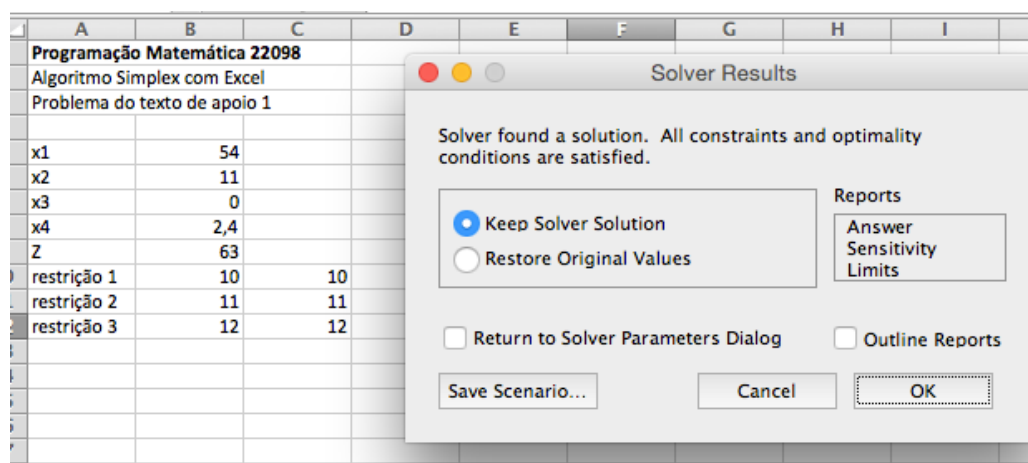
Buttons: Add, Cancel, OK

Programação Matemática 22098			
Algoritmo Simplex com Excel			
Problema do texto de apoio 1			
x1	0		
x2	0		
x3	0		
x4	0		
Z	0		
restrição 1	0	10	
restrição 2	0	11	
restrição 3	0	12	

Nesta janela, em **“Cell Reference:”** indicamos a célula onde programamos a primeira restrição, escolhemos o tipo (maior, menor, igual) e em **“Constraint:”** ou colocamos ou o valor à mão ou a célula onde esse valor está. Com isto feito clicamos em **“Add”** e passamos para a próxima. Inseridas as três restrições, clicamos em **“OK”**.

5. Estamos por fim em condições de mandar resolver. Se o problema exigir solução positiva, é clicar na checkbox de **“Make Unconstrained Variables Non-negative”** na janela *Solver Parameters*. Por último, em **“Select a Solving Method”** escolhe-se *simplex LP* e clica-se **“Solve”**.

6. Após alguns segundos o Excel vai devolver a mensagem seguinte:



A mensagem **“Solver found a solution (...)”** significa que a resolução foi bem-sucedida. Outra mensagem no mesmo local normalmente quererá dizer que o algoritmo encontrou uma situação particular, como as descritas no capítulo 2.7 do livro de texto de JCR.

Clica-se no **“Keep Solver Solution”** e onde estava o ponto inicial temos agora a solução ótima procurada.

Analisando a solução encontrada, vemos que as restrições saturaram todas, pelo que não há folgas nenhuma nas variáveis de desvio. Todos os recursos estão a ser utilizados de forma ótima, sem desperdício. C.f. livro de texto de JCR, p.83.

## Programação inteira e não-linear com o solver do Excel

O solver do Excel também permite resolver problemas de otimização para além dos lineares. No entanto, para outros tipos de problema, o solver **não garante** que o ótimo obtido seja global. Além disso, este ótimo vai depender, por vezes fortemente, do ponto de partida que se der ao iniciar o algoritmo (i.e. valores iniciais das variáveis de decisão).

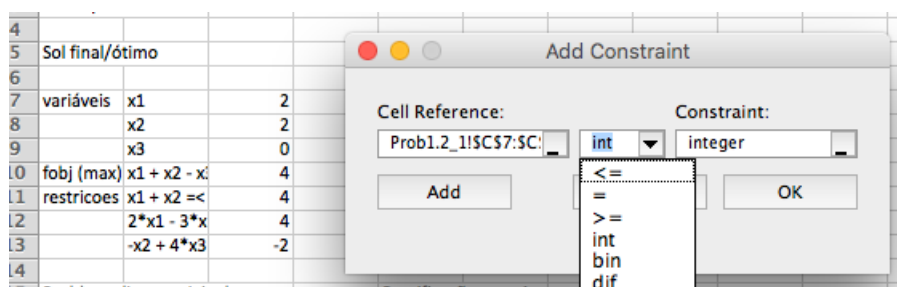
### Programação inteira

Aqui o termo “programação inteira” refere-se ao caso em que o problema é linear na função objetivo e restrições, mas as variáveis de decisão são todas restringidas a valores inteiros. Na verdade, o termo correto é “programação linear inteira” (nem todos os problemas inteiros são lineares). Problemas lineares com variáveis reais e inteiras à mistura dizem-se de “programação linear mista”, e o solver também pode ser usado para resolver este caso.

No caso da programação inteira (e mista) é extremamente fácil parametrizar o solver para resolver o problema. Basta formular o problema como um problema linear normal e adicionar restrições de integralidade para as variáveis em questão.

Para tal basta, no menu **Add Constraint**, adicionar uma nova restrição, seleccionar as variáveis a restringir a inteiros e escolher na coluna do meio “**int**” ou “**bin**”, conforme se queira que a variável tome valores inteiros ou binários (valor 0 ou 1, adequado para casos do tipo sim/não).

Exemplo:



Aqui coloca-se a restrição de que as variáveis nos campos x1 a x3 são inteiras.

Depois basta correr o solver como usualmente. Este resolve o problema inteiro usando o algoritmo *branch and bound*, que basicamente vai dividindo o espaço de soluções em pedaços sucessivos até encontrar as soluções ótimas inteiras (c.f. livro de texto JCR, cap.6). Dado que processo recorre ao simplex para cada ramo, todo o processo é bem mais lento do que o problema linear, podendo chegar a minutos, ou mesmo horas, de espera para problemas de dimensão média/grande. Para problemas pequenos é relativamente rápido.

Em todo caso convém não esquecer a ressalva de que nem sempre o ótimo encontrado é global...

Caso o problema esteja a demorar muito tempo a resolver, o Solver tem a opção de tentar encontrar mais rapidamente um ótimo, mas este será apenas um ótimo aproximado. Para tal basta escolher em “**Select a Solving Method**” a opção *Evolutionary* (em vez de *simplex LP*). Esta opção recorre a algoritmos especiais, ditos *genéticos*, que trocam a exatidão da solução pela rapidez de procura. Estes algoritmos usam de aleatoriedade na procura, pelo que nem sempre darão o mesmo resultado. Nas opções de *Evolutionary* pode-se impor um tempo máximo para a corrida.



## Programação não-linear

O solver também resolve problemas não-lineares. Trata-se de problemas em que há pelo menos uma função objetivo ou restrição não-linear nas variáveis de decisão. P.ex.

$$\text{Min } Z = x_1^2 + 4x_2$$

$$\text{s. a. } 2x_1 - x_2 \leq 5$$

$$x_1 + x_2^3 \leq 1$$

$$x_1, x_2 \geq 0$$

é não-linear na função objetivo e na 2ª restrição.

Problemas não-lineares podem ser resolvidos pelo solver, bastando para isso usar em ***“Select a Solving Method”*** a opção *GRGNonLinear*.

Tal como no caso inteiro e misto, o ótimo não-linear depende do ponto inicial de procura e pode não ser global.

## Programação linear, inteira e mista – o lpSolve

O lpSolve é um programa feito em linguagem C para resolver problemas de programação linear, inteira e mista, de uso gratuito. É pois, tal como o Excel, um solver. O lpSolve é no entanto apenas um pedaço de código e não tem nenhum interface gráfico como o Excel (GUI – *graphical user interface*). Para usar o lpSolve há duas formas: ou se prepara um ficheiro de input e se manda correr ou se dá os comandos que definem o modelo um-a-um via um programa que interaja com o lpSolve. Um tal programa é o software estatístico “R” (entre dezenas de outros!), que além de correr o lpSolve é uma poderosa ferramenta *open source* e grátis de cálculo estatístico.

O interface R é mais interativo do que correr o lpSolve sobre ficheiros de input. Requer, no entanto, instalar-se dois programas (embora se possa instalar o lpSolve a partir do R).

Abaixo vamos explorar as duas formas de usar o lpSolve. O leitor depois escolherá a que preferir. Para quem não usa o R, é talvez preferível usar o lpSolve diretamente. Para quem está habituado ao R, pode ser mais fácil puxar a package lpSolve e trabalhar no ambiente que já conhece.

Uma limitação do lpSolve é que não usa múltiplos processadores. Solvers comerciais, como p.ex. o CPLEX, fazem uso de todo o poder de processamento da máquina em que correm.

### lpSolve em ambiente R

#### Instalando o R

O R pode ser puxado em <https://www.r-project.org/> a instalação, seja em Windows, MacOS ou Linux, é relativamente simples, bastando escolher a versão e seguir as instruções no ecrã. Se porventura tiver dificuldades, contacte o professor.

#### Instalando o lpSolve no R

Como é normal em projetos *open source* em que qualquer utilizador pode adicionar código, o R vem apenas com o código base e funciona por módulos suplementares. Há milhares destes módulos para o R, um deles sendo o lpSolve. Para instalar o lpSolve basta ir aos menus *package installer* do R e procurar por “lpSolve”. Também aqui a adição do módulo lpSolve é relativamente simples.

Ah... depois de instalar o módulo lpSolve é ainda preciso ativá-lo. Para isso é ir aos menus do R e procurar *load package* e escolher lpSolve.

#### Usando o lpSolve no R

Por esta altura o leitor já se terá apercebido que o R funciona por linha de comandos. Ou seja, todos os comandos são dados diretamente por escrita e não com cliques de rato. A forma mais simples de explicar quais são os comandos do lpSolve é com um exemplo. Consideremos p.ex. o problema linear genérico que resolvemos no solver do Excel:

$$\text{Max } Z = 2x_1 - 3x_2 + 4x_3 - 5x_4$$

$$\text{s. a. } x_1 - 4x_2 \leq 10$$

$$x_2 + 7x_3 \leq 11$$

$$x_3 + 5x_4 \geq 12$$

$$x_1, x_2, x_3, x_4 \geq 0$$

A sequência de comandos R para definir e resolver este problema são, juntamente com a respetiva explicação,

```
> probl<-make.lp(0,4)
```

Cria um problema linear, de nome “probl”, com 0 restrições e 4 variáveis de decisão. Não se preocupe com o “0” no n.º de restrições porque elas serão adicionadas uma-a-uma.

```
> set.objfn(probl,c(2,-3,4,-5))
```

Define a função objetivo do problema de nome “probl”, cujos coeficientes nas variáveis de decisão são a lista (o “c” significa lista) 2, -3, 4, -5. Isto reproduz em linguagem lpSolve a definição  $Z = 2x_1 - 3x_2 + 4x_3 - 5x_4$ .

```
> add.constraint(probl,c(1,-4,0,0), "<=", 10)
```

```
> add.constraint(probl,c(0,1,7,0), "<=", 11)
```

```
> add.constraint(probl,c(0,0,1,5), ">=", 12)
```

Define as três restrições, indicando: o nome do problema; a lista de coeficientes de cada variável de decisão; a indicação do sentido da restrição; e o valor limite. As restrições  $x_1, x_2, x_3, x_4 \geq 0$  estão assumidas por omissão (*default*) e não é preciso indicá-las.

```
> lp.control(probl,sense="max")
```

Indica que o sentido da otimização do problema “probl” é maximizar. Se não dissermos nada, ele vai minimizar por omissão. Correr este comando vai fazer imprimir no ecrã uma série de coisas, mas isso agora não interessa.

```
> probl
```

```
Model name:
```

	C1	C2	C3	C4		
Maximize	2	-3	4	-5		
R1	1	-4	0	0	<=	10
R2	0	1	7	0	<=	11
R3	0	0	1	5	>=	12
Kind	Std	Std	Std	Std		
Type	Real	Real	Real	Real		
Upper	Inf	Inf	Inf	Inf		
Lower	0	0	0	0		

Este comando não é estritamente necessário. Serve apenas para visualizar/verificar qual é a definição do problema. Assim podemos ver se nos enganámos algures na entrada de dados.

Nota 1: se nos enganarmos numa restrição, ver qual é a linha (digamos que é a R3) e correr

```
> delete.constraint(prob1, 3)
```

e em seguida voltar a introduzir a linha corrigida com `add.constraint()`.

Nota 2: por vezes pode ser preciso alterar  $x_1, x_2, x_3, x_4 \geq 0$ . Se quisermos p.ex. dizer que  $x_1, x_2, x_3, x_4 \geq (-1, -2, -3, -4)$  há que correr o comando

```
> set.bounds(prob1, lower = c(-1, -2, -3, -4))
```

```
> solve(prob1)
[1] 0
```

Manda resolver o problema linear “prob1”. O resultado “0” significa que foi encontrada uma solução ótima. Para saber qual foi essa solução, temos os comandos

```
> get.objective(prob1)
[1] 63
> get.variables(prob1)
[1] 54.0 11.0 0.0 2.4
> get.constraints(prob1)
[1] 10 11 12
```

que imprimem no ecrã os valores da função objetivo, das variáveis de decisão e o status das restrições para a solução ótima de “prob1” encontrada.

A solução é a mesma do solver Excel, como esperado.

## Programação inteira e mista com lpSolve em ambiente R

Podemos alterar o carácter das variáveis dos problemas lpSolve muito simplesmente. P.ex. se quisermos passar a variável 4 de “prob1” de real a inteira basta correr

```
> set.type(prob1, 4, type="integer")
```

Isto indica que o tipo da variável n.º 4 de “prob1” deve ser inteira. As outras permanecem reais, configurando-se assim um problema de programação linear mista:

```
> prob1
Model name:
      C1      C2      C3      C4
Maximize  2      -3      4      -5
R1         1      -4      0      0  <=  10
R2         0       1      7      0  <=  11
R3         0       0      1      5  >=  12
Kind      Std     Std     Std     Std
Type      Real   Real   Real   Int
Upper     Inf    Inf    Inf    Inf
Lower     0      0      0      0
```

Resolvendo este modelo obtemos

```
> solve(prob1)
[1] 0
> get.objective(prob1)
[1] 60
> get.variables(prob1)
[1] 54 11 0 3
```

O comando `set.type()` permite também definir uma variável como binária, i.e. a variável só pode ter valores 0 ou 1. Exemplo: (passar  $x_2$  a binária)

```
> set.type(prob1, 2, type="binary")
> prob1
Model name:
```

	C1	C2	C3	C4		
Maximize	2	-3	4	-5		
R1	1	-4	0	0	<=	10
R2	0	1	7	0	<=	11
R3	0	0	1	5	>=	12
Kind	Std	Std	Std	Std		
Type	Real	<b>Int</b>	Real	Int		
Upper	Inf	<b>1</b>	Inf	Inf		
Lower	0	0	0	0		

Note-se que “binary” é implementado passando a inteiro  $x_2$  entre 0 e 1 (lower e upper bounds).

Por curiosidade aqui fica o resultado ótimo:

```
> solve(prob1)
[1] 0
> get.objective(prob1)
[1] 15.71429
> get.variables(prob1)
[1] 14.000000 1.000000 1.428571 3.000000
```

## lpSolve com ficheiro de input

Para correr o lpSolve diretamente com um ficheiro de input é preciso, em primeiro lugar, instalar o software. Este pode ser obtido gratuitamente de

<http://lpsolve.sourceforge.net/5.5/>

O link “download” na barra esquerda leva-nos ao servidor de descarga do Sourceforge. Clicar no link [See All Activities](#) no fundo página de descarga e puxar: (use “find” na página)

<b>Windows:</b>	/lpsolve/5.5.2.3/lp_solve_5.5.2.3_exe_win32.zip	(computadores 32-bit)
	/lpsolve/5.5.2.3/lp_solve_5.5.2.3_exe_win64.zip	(computadores 64-bit)
<b>Linux :</b>	/lpsolve/5.5.2.3/lp_solve_5.5.2.3_exe_ux32.tar.gz	(computadores 32-bit)
	/lpsolve/5.5.2.3/lp_solve_5.5.2.3_exe_ux64.tar.gz	(computadores 64-bit)
<b>MacOS:</b>	/lpsolve/5.5.2.3/lp_solve_5.5.2.3_exe_osx32.tar.gz	(só disponível 32-bit)

Notas:

Os números de versão “5.5.2.3” poderão ter mudado à altura da consulta da página...

As versões 32-bit funcionam em computadores 64-bit. O contrário não. As versões 64-bit são mais rápidas, mas para problemas pequenos a diferença é impercetível.

As versões com “/working” no path são versões para testes e podem ter bugs. Não se recomenda puxar essas.

Depois de puxar, descompactar (unzip, gzip) e instalar num diretório. Basta ter os ficheiros nesse diretório para poder correr o lpSolve. À partida as instruções de instalação no ecrã chegarão... Se não chegarem, contactar o professor.

## Correndo o lpSolve

Para correr o lpSolve há preparar o ficheiro de input, que pode ser um mero ficheiro de texto feito num editor de texto, p.ex. Notepad do Windows, TextEdit do Mac, etc. Normalmente este ficheiro é guardado com extensão .LP, para se identificar facilmente o que é, mas pode também ser gravado com .TXT.

Vejamos um exemplo do aspeto de um ficheiro .LP:

```
/* model.lp */  
  
max: 143 x + 60 y;  
  
120 x + 210 y <= 15000;  
110 x + 30 y <= 4000;  
x + y <= 65;
```

A sintaxe é autoexplicativa, bem como o problema que define. A 1ª linha é simplesmente de comentários e note-se o ponto-e-vírgula, que age como separador de expressões e que é fulcral na sintaxe.

Para resolver basta colocar o ficheiro .LP no mesmo diretório onde estão instalados os ficheiros do lpSolve, abrir uma janela de DOS (Windows) ou shell Unix (Linux/Mac), ir ao diretório e escrever:

```
> lp_solve.exe testel.lp      (Windows)
> ./lp_solve testel.lp      (Linux/Mac)
```

o que devolverá o output:

```
Value of objective function: 6026.87500000
```

```
Actual values of the variables:
```

```
x                25.625
y                39.375
```

Se quisermos evitar proliferação de ficheiros no diretório principal do lpSolve podemos criar um subdiretório, p.ex. testes, gravar lá os ficheiros e correr, a partir do diretório do lpSolve,

```
> lp_solve.exe testes\testel.lp  (Windows)
> ./lp_solve testes/testel.lp    (Linux/Mac)
```

para resolver problemas de programação inteira ou mista, basta adicionar no final do ficheiro a natureza das variáveis, *as follows*:

```
/* model.lp */

max: 143 x + 60 y;

120 x + 210 y <= 15000;
110 x + 30 y <= 4000;
x + y <= 65;

int x;
bin y;
```

Esta alteração faz o lpSolve mudar as rotinas de cálculo para programação inteira *branch-and-bound* e devolverá

Value of objective function: 5208.00000000

Actual values of the variables:

x	36
y	1

## Modo verboso

Para problemas que levem algum tempo a resolver, o lpSolve tem um modo que permite imprimir informação no ecrã (*verbose mode*). Assim, o utilizador pode ir seguindo a evolução da resolução do problema e avaliar quanto tempo restará até ao final dos cálculos.

Para ativar o modo verboso, basta incluir na linha-de-comandos a flag `-vX`, em que X é um valor entre 0 e 6, dependendo da quantidade de informação que se quer. Usualmente `-v4` é bom compromisso entre visibilidade e informação.

Vejamos o exemplo acima em modo verboso:

```
> lp_solve testel.lp -v4
```

```
Model name: '' - run #1
Objective:  Maximize(R0)
```

```
SUBMITTED
Model size:      3 constraints,      2 variables,      6 non-zeros.
Sets:           0 GUB,              0 SOS.
```

```
Using DUAL simplex for phase 1 and PRIMAL simplex for phase 2.
The primal and dual simplex pricing strategy set to 'Devex'.
```

```
Relaxed solution      5221 after      1 iter is B&B base.
Feasible solution     5208 after      3 iter,          2 nodes (gap 0.2%)
Optimal solution      5208 after      3 iter,          2 nodes (gap 0.2%).
```

```
Relative numeric accuracy ||*|| = 0
```

```
MEMO: lp_solve version 5.5.2.5 for 32 bit OS, with 64 bit REAL variables.
In the total iteration count 3, 1 (33.3%) were bound flips.
There were 1 refactorizations, 0 triggered by time and 0 by density.
... on average 2.0 major pivots per refactorization.
The largest [LUSOL v2.2.1.0] fact(B) had 7 NZ entries, 1.0x largest basis.
The maximum B&B level was 2, 0.5x MIP order, 2 at the optimal solution.
The constraint matrix inf-norm is 210, with a dynamic range of 210.
Time to load data was 0.000 seconds, presolve used 0.000 seconds,
... 0.000 seconds in simplex solver, in total 0.000 seconds.
```

Value of objective function: 5208.00000000

Actual values of the variables:

x	36
y	1

E agora um exemplo mais complicado, só com binárias. O ficheiro de input está abaixo. Experimente fazer copy-paste deste para um ficheiro e corra o problema no seu terminal.

```
/* model.lp */
```

```
min: 2.55 - 1.275 x11 - 0.6375 B1 + 2.43 - 1.215 x12 - 0.6075 B1 + 1.45 - 0.725 x13 - 0.3625 B1 +
2.83 - 1.415 x21 - 0.7075 B2 + 2.71 - 1.355 x22 - 0.6775 B2 + 1.5 - 0.75 x23 - 0.375 B2 + 1.78 - 0.89
x31 - 0.445 B3 + 2.07 - 1.035 x32 - 0.5175 B3 + 2.02 - 1.01 x33 - 0.505 B3 + 2.12 - 1.06 x41 - 0.53
```



```

B4 + 2.48 - 1.24 x42 - 0.62 B4 + 1.3 - 0.65 x43 - 0.325 B4 + 2.52 - 1.26 x51 - 0.63 B5 + 1.8 - 0.9
x52 - 0.45 B5 + 1.46 - 0.73 x53 - 0.365 B5 ;

B1 >= x11 + x12 + x13 - 2 ;
B2 >= x21 + x22 + x23 - 2 ;
B3 >= x31 + x32 + x33 - 2 ;
B4 >= x41 + x42 + x43 - 2 ;
B5 >= x51 + x52 + x53 - 2 ;
3 B1 <= x11 + x12 + x13 ;
3 B2 <= x21 + x22 + x23 ;
3 B3 <= x31 + x32 + x33 ;
3 B4 <= x41 + x42 + x43 ;
3 B5 <= x51 + x52 + x53 ;
2979.4 x11 + 2277.8 x12 + 1822.0 x13 + 2019.5 x21 + 1940.1 x22 + 1248.5 x23 + 1306.4 x31 + 2174.5 x32
+ 1779.7 x33 + 2576.8 x41 + 2138.5 x42 + 1034.2 x43 + 3237.4 x51 + 1766.6 x52 + 1461.1 x53 <= 15000
;
bin x11 ;
bin x12 ;
bin x13 ;
bin x21 ;
bin x22 ;
bin x23 ;
bin x31 ;
bin x32 ;
bin x33 ;
bin x41 ;
bin x42 ;
bin x43 ;
bin x51 ;
bin x52 ;
bin x53 ;
bin B1 ;
bin B2 ;
bin B3 ;
bin B4 ;
bin B5 ;

```

## Correndo o problema, temos...

```
> lp_solve testeXPTO.lp -v4
```

```

Model name:  '' - run #1
Objective:   Minimize(R0)

```

SUBMITTED

```

Model size:      11 constraints,      20 variables,      55 non-zeros.
Sets:            0 GUB,              0 SOS.

```

Using DUAL simplex for phase 1 and PRIMAL simplex for phase 2.  
The primal and dual simplex pricing strategy set to 'Devex'.

```

Relaxed solution      17.1602539731 after      28 iter is B&B base.

Feasible solution      20.44 after      76 iter,      25 nodes (gap 18.1%)
Improved solution      20.415 after     91 iter,      36 nodes (gap 17.9%)
Improved solution      20.395 after     95 iter,      37 nodes (gap 17.8%)
Improved solution      20.38 after    128 iter,      64 nodes (gap 17.7%)
Improved solution      20.3575 after    169 iter,      92 nodes (gap 17.6%)
Improved solution      19.6775 after    267 iter,     144 nodes (gap 13.9%)
Improved solution      19.5375 after    340 iter,     186 nodes (gap 13.1%)
Improved solution      19.3775 after    358 iter,     195 nodes (gap 12.2%)
Improved solution      19.27 after     392 iter,     215 nodes (gap 11.6%)
Improved solution      18.8825 after    592 iter,     333 nodes (gap 9.5%)

Optimal solution      18.8825 after    1019 iter,     598 nodes (gap 9.5%).

```

Relative numeric accuracy ||\*|| = 0

```

MEMO: lp_solve version 5.5.2.5 for 32 bit OS, with 64 bit REAL variables.
      In the total iteration count 1019, 250 (24.5%) were bound flips.
      There were 299 refactorizations, 0 triggered by time and 0 by density.
      ... on average 2.6 major pivots per refactorization.

```

```
The largest [LUSOL v2.2.1.0] fact(B) had 29 NZ entries, 1.0x largest basis.  
The maximum B&B level was 15, 0.4x MIP order, 11 at the optimal solution.  
The constraint matrix inf-norm is 3237.4, with a dynamic range of 3237.4.  
Time to load data was 0.000 seconds, presolve used 0.000 seconds,  
... 0.000 seconds in simplex solver, in total 0.000 seconds.
```

Value of objective function: 18.88250000

Actual values of the variables:

x11	0
B1	0
x12	1
x13	0
x21	1
B2	1
x22	1
x23	1
x31	1
B3	1
x32	1
x33	1
x41	0
B4	0
x42	1
x43	0
x51	0
B5	0
x52	0
x53	0

Como se pode ver do indicado a vermelho, o lpSolve executou 1019 iterações de branch-and-bound neste problema de 20 binárias e 10 restrições, sendo que o ótimo com variáveis binárias tem uma diferença (*gap*) de 9,5% para com o ótimo real.

Em <http://lpsolve.sourceforge.net/5.5/> encontrarão mais detalhes sobre coisas que se podem fazer com o lpSolve, inclusive como o correr diretamente a partir do sistema operativo, etc.

E pronto era isto. Happy solving! ☺