Universidade Estadual de Campinas
IMECC - Instituto de Matemática, Estatística e Computação Científica

Projeto Supervisionado II

# Tutorial sobre modelagem em VBA

Aluna: Tatiana Suemy Otsuka
Orientador: Prof. Dr. Antonio Carlos Moretti – Dpto. de matemática aplicada

5 de Julho de 2012

# Índice

# Objetivo

O presente projeto tem como objetivo servir de um breve tutorial para aqueles interessados em utilizar a plataforma Microsoft Excel e a linguagem, Visual Basic for Applications (VBA) como ferramentas para a modelagem de problemas de otimização. Procura-se com este tutorial poder exemplificar a funcionalidade dessa ferramenta para modelos de suporte de decisão.

# Introdução

O propósito deste trabalho não é ensinar em detalhes a linguagem VBA, nem desenvolver a modelagem de um novo problema, mas sim ser um guia para iniciar a construção de modelos fazendo uso dessa linguagem e explorando seus diferenciais e praticidades.

Através do VBA, modelos relativamente complexos podem ser acessíveis a qualquer usuário de Excel, sendo apresentados de forma amigável através de simplificadas telas de entrada de dados e claros relatórios finais.

A primeira parte deste tutorial consiste em uma breve explicação das principais ferramentas, conceitos e comandos utilizados nos modelos. Em seguida é desenvolvido um modelo de transporte clássico, aplicando o que foi mostrado primeiramente.

Pré-requisitos: Este tutorial assume que os leitores estejam familiarizados com modelos clássicos de pesquisa operacional, que tenham experiência com alguma linguagem de programação e sejam usuários da interface do Excel.

Observação: A construção deste tutorial foi baseada na versão do Excel 2007, contudo para as outras versões há poucas mudanças na linguagem VBA em si.

# 1. O que é VBA?

VBA (Visual Basic for Applications) é uma linguagem de programação com elementos típicos, como loopings, construções de if-then-else, vetores, funções, etc.

O “for applications” do nome VBA, significa que qualquer software de aplicação, como Excel, Access, Word, ou até mesmo pacotes que não sejam da Microsoft, podem ser manipulados programando-se em VBA. A diferença básica entre eles consiste em seus objetos específicos. Por exemplo, alguns objetos do Excel são células e planilhas. Já o Word possui parágrafos, e o PowerPoint Slides.

Todos que possuem o Microsoft Office instalado no computador, podem utilizar o VBA. No Excel, o **Visual Basic Editor (VBE),** que é a área onde é feita a programação, pode ser acessado pressionando-se **Alt+F11**.

# 2. Objetos do Excel

Para exemplificar um objeto do Excel, considere uma única célula. Essa célula é um objeto do tipo **Range**, que tem propriedades, como a propriedade **Value** (texto ou número digitado na célula). Esse objeto tem também métodos, por exemplo, o método **Copy** (função que copia o Range), que possui argumentos como o **Destination** (Local aonde será colado o que foi copiado).

Alguns objetos do Excel possuem eventos. Por exemplo, o objeto **Workbook** possui o evento **Open**.

Também existem as coleções de objetos que similarmente aos objetos, possuem métodos e propriedades, porém estes nem sempre são os mesmos dos objetos que contêm. Coleções são diferenciadas por serem nomeadas no plural. Por exemplo, a coleção de objetos **Worksheets**, é a coleção de todas as planilhas do arquivo, e cada planilha é um objeto do tipo **Worksheet**.

Uma ótima ferramenta para verificar todos os objetos, suas propriedades, métodos, eventos é o **Pesquisador de objeto** que pode ser acessado dentro do editor VBE em **Exibir -> Pesquisador de objeto**.

Outra ferramenta de ajuda que sempre deve ser utilizada é o **Intellisense**. Ao digitar em um Módulo, o nome de um objeto, seguido de um ponto, todas as propriedades e métodos desse objeto serão listados, veja figura 2.1. Essa ferrmamenta também lista todos os argumentos de um método ao digitar o método e pressionar a tecla de espaço, como mostra a figura 2.2. Qualquer argumento exibido em chaves é opcional, os outros são obrigatórios.

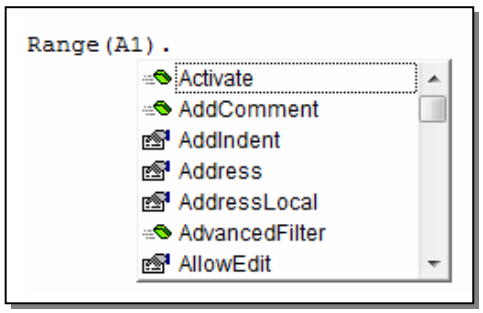



Figura 2.1

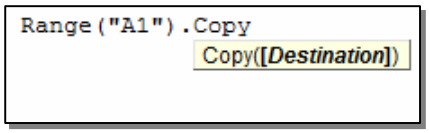



Figura 2.2

# 3. Conceitos básicos

Esse capítulo pretende mostrar de forma breve como é a sintaxe dos comandos necessários para o entendimento da modelagem que será feita no capítulo ().

## 3.1 Sub-rotinas

Uma modelagem geralmente possui muitas tarefas, o que em VBA é chamado de **sub-rotina, macro** ou simplesmente **sub.** Uma sub geralmente é um grupo de códigos digitados em um **Módulo**, que executam uma determinada tarefa. Para inserir um módulo entre no editor VBE em **(Inserir -> Módulo)**. Uma sub-rotina tem um nome e pode ou não ter argumentos como em:

```
Sub Teste(Name As String, Number As Integer)
…
End Sub
```

Para exemplificar, esta sub-rotina pode ser chamada da seguinte forma:

```
Sub Chamar()
Dim Nome As Sting, Numero As Integer
        ...
        Call Teste(Nome, Num)
End Sub
```

Ao chamar uma sub-rotina com argumentos, eles devem coincidir em tipo, ordem e número, mas não precisam ter os mesmo nomes.

## 3.2 Variáveis e Constantes

Como em muitas linguagens, as variáveis precisam ser declaradas. Isso pode ser feito através da sentença **Dim**. Os tipos de variáveis mais utilizados são:

- **String** (para palavras);
- **Integer** (para valores inteiros);
- **Long** (para valores inteiros grandes);

- **Boolean** (assume os valores True ou False);
- **Single** (para números decimais);
- **Double** (para números decimais com mais casas);
- **Currency** (para valores monetários);
- **Variant** (pode ser de qualquer tipo a ser decidido pelo VBA).

Exemplo:

```
Sub Var()
        Dim j As Integer
        Dim name As String
        Dim Test As Boolean
End Sub
```

Ou, simplificando:

```
Sub Var()
        Dim j As Integer, name As String, Test As Boolean
End Sub
```

**Option Explicit:** Utilizando o Comando **Option Explicit** no início de um módulo forçamos a declaração das variáveis, antes de qualquer uso dela. Se alguma não for declarada, uma mensagem de erro aparecerá na tela.

- **Objetos:**

Um objeto também é uma variável e deve ser declarado como em:

```
Dim celula As Range
```

Fazendo isso, podemos atribuir um range a "celula" e trabalhar com essa variável.

Exemplo:

```
Dim celula As Range
Set celula = ActiveWorkbook.Worksheets("Data").Range("Nomes")
celula.Font.Size=12
```

O comando Set sempre é usado para atribuir valor ou definir uma variável do tipo Objeto, mas nunca é usado para outros tipos de variáveis. Por exemplo, para uma variável do tipo "Integer", atribuímos valor desta forma:

```
Dim j As Integer
j=20
```

- **Constantes**

Uma constante é declarada da forma:

```
Const Taxa=0.9
```

- **Declarando variáveis públicas**

Se o programador declarar uma variável dentro de certa sub-rotina, esta variável será local, ou seja, será acessada apenas por essa sub-rotina. Caso deseje-se utilizar uma variável em mais de uma sub-rotina do módulo, é necessário declarar a variável da forma como foi explicado, porém no início do módulo, antes de qualquer sub-rotina. Agora, se o modelo tiver mais de um módulo, ou, por exemplo, um Event Handler para UserForms, e deseja-se que certa variável seja visível para todos os módulos e Event Handlers, então declara-se a variável também no início do módulo mas da seguinte forma:

```
Public variavel As Integer
```

Pelo padrão do VBA, sub-rotinas são públicas, ou seja, podem ser chamadas por qualquer outra sub-rotina do projeto. Para que essa sub-rotina só possa ser chamada dentro de um módulo, a declaramos desta forma:

```
Private Sub Test()
```

## 3.3 Caixas de Entrada e Mensagens

Duas das mais comuns tarefas de programas em VBA são: pegar dados do usuário e mostrar mensagens ou resultados. Para isso existem as funções **InputBox** e **MsgBox,** respectivamente.

O InputBox recebe pelo menos um argumento, uma mensagem informando o usuário o que se espera que ele digite como entrada. Um segundo argumento que pode ser usado é o título da caixa.

Exemplo:

```
Dim Cor As String
Cor = InputBox("Entre com a cor desejada", "Escolhendo a cor")
```

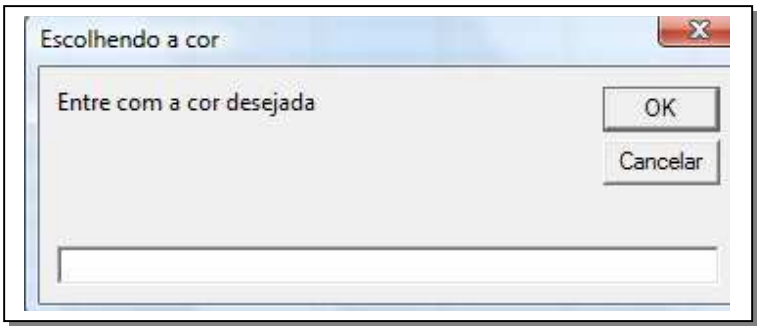



Figura 3.1

O MsgBox recebe pelo menos um argumento, a mensagem que deseja-se informar. Dois argumentos opcionais são: um botão de indicação e o título da caixa.

Exemplo:

```
MsgBox "A cor é laranja.", vbInformation, "Cor escolhida"
```

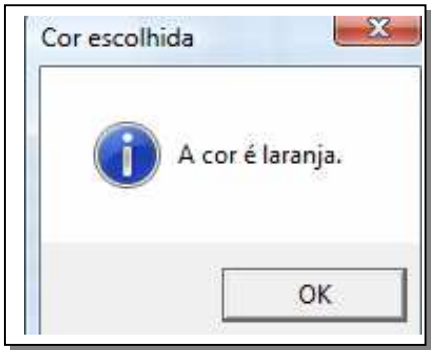



Figura 3.2

## 3.4 Comentários, Concatenação de String e Pular Linhas

Segue abaixo as sentenças desses 3 tópicos:

### Comentários

Coloca-se o caracter " ' " antes da linha a ser comentada.

Exemplo:

```
' Esta linha é um comentário
```

### Pular linhas

Quando uma linha de código exceder o tamanho da tela, é possível cortá-la e continuar na próxima linha através dos caracteres (espaço + underline). Por exemplo:

```
MsgBox InputBox("Digite um número a ser exibido na caixa de mensagem", _
"Entrada de dados"), vbInformation, "Número digitado"
```

## Concatenação de Strings

Quando quereremos escrever uma mensagem com partes literais e outras variáveis, por exemplo, o valor de uma variável, devemos concatenar uma string, separando as partes com o caractere "**&**":

Exemplo:

```
Dim product As Integer
product=12
MsgBox "O número do produto é" & product & "."
```

# 3.5 Vetores

Declara-se um vetor desta forma:

```
Dim vetor(100) As String
```

Quando não se sabe de antemão o tamanho do vetor, o declaramos desta forma:

```
Dim vetor() As String
```

Se o tamanho do vetor for conhecido ao longo da execução do programa, deve-se redimensioná-lo para o armazenamento necessário de memória. Para isso, utiliza-se a sentença **Redim:**

```
numero= InputBox("Quantos índices o vetor terá?")
Redim vetor(Numero)
```

Pode-se usar a declaração Redim toda vez que se deseja mudar a dimensão de um vetor. Porém ao fazer isso, todo o conteúdo do vetor será apagado. Para que isso não ocorra, é necessário utilizar a declaração **Preserve**:

```
numero=numero+1
Redim Preserve vetor(1 to Numero)
```

## A declaração “Option Base”

O padrão do VBA para o primeiro índice de um vetor é 0 e não 1, o que é chamado de “0-base indexing”.

Para que o primeiro índice seja 1, é necessário acrescentar a seguinte linha no início de um módulo:

```
Option Base 1
```

Também pode-se forçar o primeiro índice de um vetor a ser 1 mesmo que o sistema esteja no padrão “0-base indexing”, declarando-o desta forma:

```
Dim vetor(1 to 100)
```

## 3.6 Propriedade offset

Essa propriedade recebe dois argumentos, o primeiro referente à linha e o segundo à coluna, indicando que a partir de determinada célula, move-se para a célula referente ao deslocamento definido pelos argumentos. Para ir à uma linha acima o argumento é posivito, e abaixo, negativo. Para ir à uma coluna à direita o argumento é positivo, à esquerda, negativo. Por exemplo:

```
Range(“B3”).Offset(-1,1).Select
```

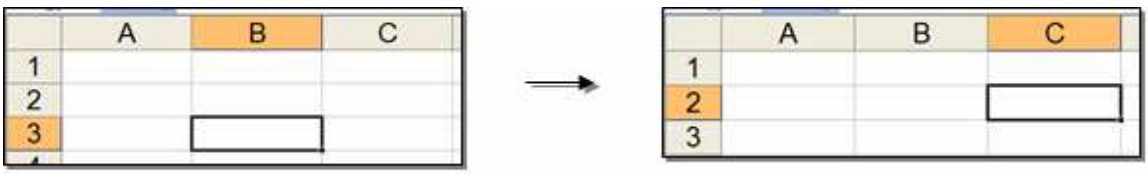



Figura 3.3

Pode-se também especificar um grupo de células, como:

```
With Range(“A1”)
        Range(.Offset(1,1), .Offset(2,2)).Select
End With
```

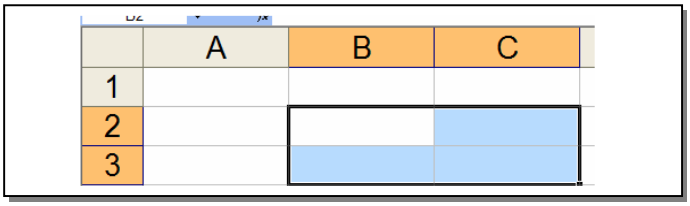


Figura 3.4

## 3.7 Propriedade End

Essa propriedade seleciona a partir de uma célula que contenha um valor, uma caixa de células que contenham valores, até encontrar uma célula vazia. Ele toma um argumento que determina uma direção. Por exemplo:

```
With Range("A1")
        Range(.Offset(0,0),.End(xlDown).End(xlToRight)).Select
End With
```

Esse comando funciona da mesma forma que selecionar a célula A1, manter pressionado a tecla Shift e pressionar a tecla End, Down e Right sucessivamente.

## 3.8 Construções com With

Este é um comando muito útil para evitar a repetição de palavras quando se trabalha com objetos, seus métodos e propriedades. Segue um exemplo da construção:

```
Workbooks("Condomínio").Worksheets("Casas").Range("A1").Value="Azul"
Workbooks("Condomínio").Worksheets("Casas"). Range("A1").Font.Name="Arial"
Workbooks("Condomínio").Worksheets("Casas"). Range("A1").Font.Bold="True"
Workbooks("Condomínio").Worksheets("Casas"). Range("A1").Font.Size="12"

'Agora usando With

With Workbooks("Condomínio").Worksheets("Casas"). Range("A1")
        .Value="Azul"
        With.Font
                .Name="Arial"
                .Bold="True"
                .Size="12"
        End With
End With
```

## 3.9 Estruturas de condições

Seguem abaixo as estruturas de condições do VBA:

### Estrutura IF

```
If condition1 Then
        Statements1
[ElseIf condition2 Then
        Statements2
…
Else
        OtherStatements]
End If
```

### Estrutura Case

```
Select Case someVariable
        Case value1
                Statements1
        Case value2
                Statements2
        …
        [Case Else
                OtherStatements]
End Select
```

## 3.10 Estruturas de Loopings

Seguem abaixo as estruturas de loopings do VBA:

### Estrutura For

```
For counter=first To last [Step increment]
        Statements
Next [counter]
```

### Estrutura For Each

```
Dim item as Object
For Each item in Collection
        Statements
Next
```

### Estruturas Do Loops

```
Do Until condition
        Statements
Loop
```

```
Do
        Statements
Loop Until condition
```

```
Do
        Statements
Loop While condition
```

```
Do While
condition
        Statements
Loop
```

# 4. Workbooks

Um **Workbook** é basicamente o arquivo em Excel., ou seja, toda a área de trabalho.

Uma coleção Workbooks é a coleção de todos os workbooks abertos. Qualquer membro dessa coleção pode ser especificado por um nome como Workbooks("Arquivo x").

Cada workbook individual possui as coleções Worksheets e Charts, por exemplo.

Outro conceito é o de que Workbooks, Worksheets e Charts também são objetos, e possuem métodos e propriedades. Uma importante propriedade é o **Count**, que conta o número de worksheets, workbooks ou charts; e um método importante é o **Add**, que adiciona uma nova coleção dentre essas.

# 5. Event Handler

Um Event Handler é uma sub-rotina que é executada sempre que certo evento ocorre. Um dos mais importantes é o seguinte:

## Workbook_Open Event Handler

Esse Event Handler responde ao evento de abrir uma pasta de trabalho, ou seja, ele ocorre ao abrir um arquivo em Excel. Ele sempre é colocado na janela ThisWorkbook ou EstaPasta_de_trabalho do VBE. Esta janela é reservada para eventos relacionados à um workbook.

# 6. User Forms

User Forms são janelas para a entrada de dados do usuário. Para construí-los é necessário elaborar a parte gráfica com a funcionalidade desejada e de forma a ser atrativa e prática para o usuário. Além disso, é necessário implementar a parte da programação que executa as tarefas determinadas pelo usuário.

## 6.1 Parte gráfica

Primeiramente, entre no editor VBE e torne visível as janelas Project Explorer (**Exibir-> Project Explorer**) e Janela de propriedades (**Exibir-> Janela de propriedades**). Para inserir um User Form clique em (**Inserir-> User Form**). Uma janela como essa será exibida:

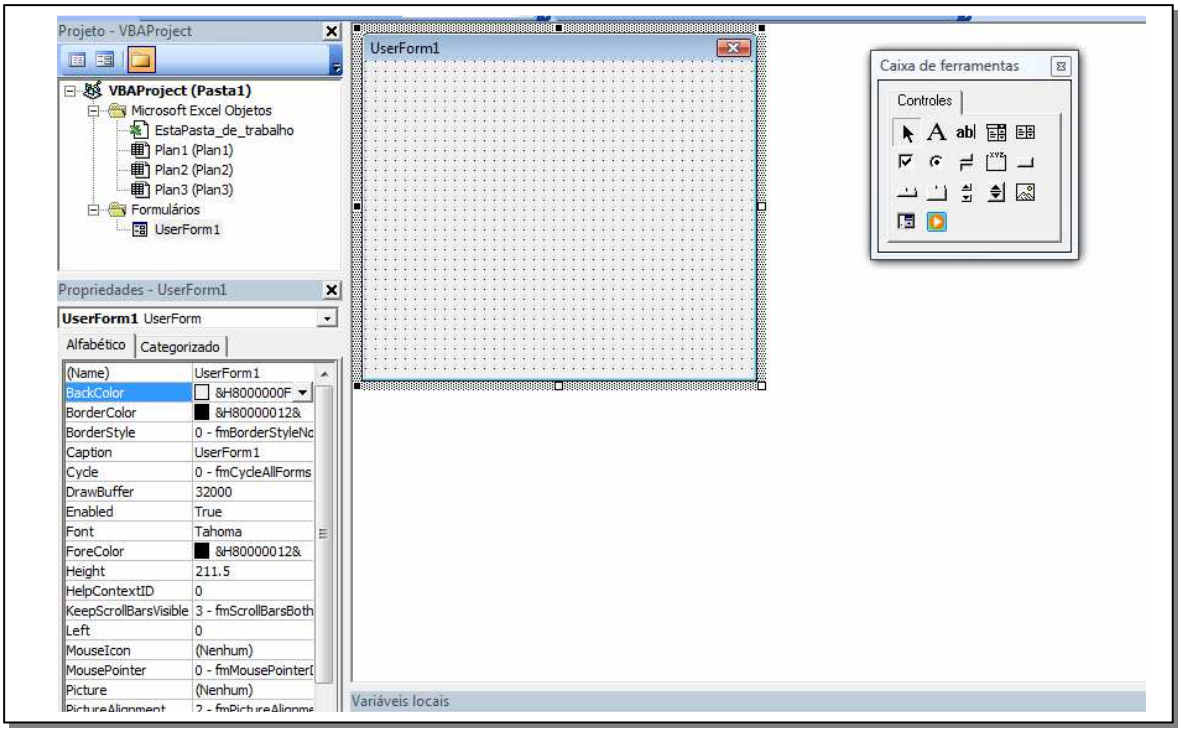



Figura 6.1

Além das ferramentas exibidas na caixa de ferramentas, outras podem ser adicionadas clicando com o botão direito na caixa e em controles adicionais.

Todas as ferramentas da caixa de ferramentas são bastante intuitivas de serem usadas, porém para explicações mais detalhadas consulte a biblioteca MSForms no Pesquisador de Objetos.

A janela de propriedades mostra as propriedades de cada controle se este estiver selecionado. Duas propriedades importantes são Name e Caption. A propriedade Name é usada como referência para um UserForm e a Caption configura os textos exibidos na tela. Essas propriedades, assim como todas as outras podem ser alteradas diretamente na janela de propriedades.

Para visualizar o resultado final do Userform clique em Executar->Executar Sub/userForm ou a tecla F5.

## 6.2 Event Handler para User Forms

Um Event Handler de uma User Form são colocados na janela de código de uma User Form. Para visualizá-la insira uma User Form como foi explicado anteriormente, e clique em **(Exibir -> código).** Nesta janela estão todas as sub-rotinas de cada item da User Form. Os nomes das sub-rotinas iniciam-se com o nome do item seguido por "underline" e um evento. Na janela de códigos, existem duas listas, uma controla os itens da User Form e a User Form em si, e a outra controla os eventos correspondentes ao que foi selecionado na primeira lista. Veja as figuras abaixo:

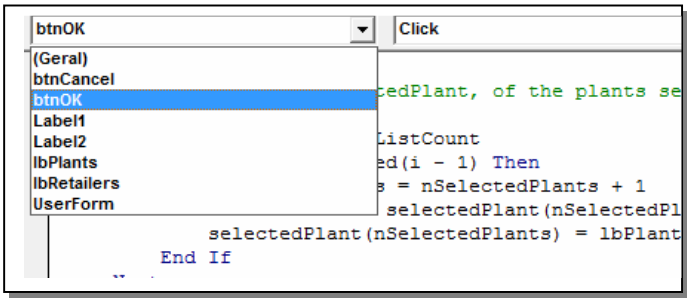


Figura 6.2

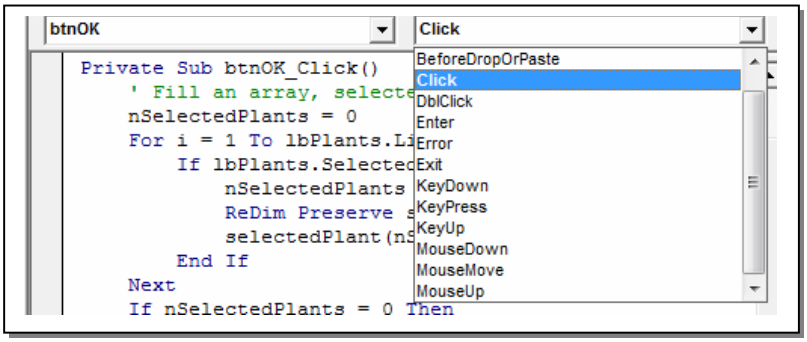


Figura 6.3

Dentro de cada sub-rotina é necessário escrever o código para a execução de determinada tarefa.

A idéia de Event Handler para User Form é decidir qual tarefa será executada como resposta a certo evento.

# 7. Chamando o SOLVER pelo VBA

Este tópico assume que o leitor saiba usar o SOLVER da forma usual através da interface do Excel. Mesmo com o uso do SOLVER pelo VBA, é necessário que as entradas e as fórmulas que relacionam os dados já estejam digitadas em uma planilha. Para informações mais detalhadas, consultar a referência [1].

Solver é um aplicativo que foi desenvolvido pela Frontline Systems, os quais felizmente escreveram várias funções em VBA, permitindo assim manipulá-lo através da programação. Para utilizar essas funções, o primeiro passo é criar uma referência para o SOLVER no VBE. Isso pode ser feito em **Ferramentas->Referências**, marcando SOLVER na lista exibida:

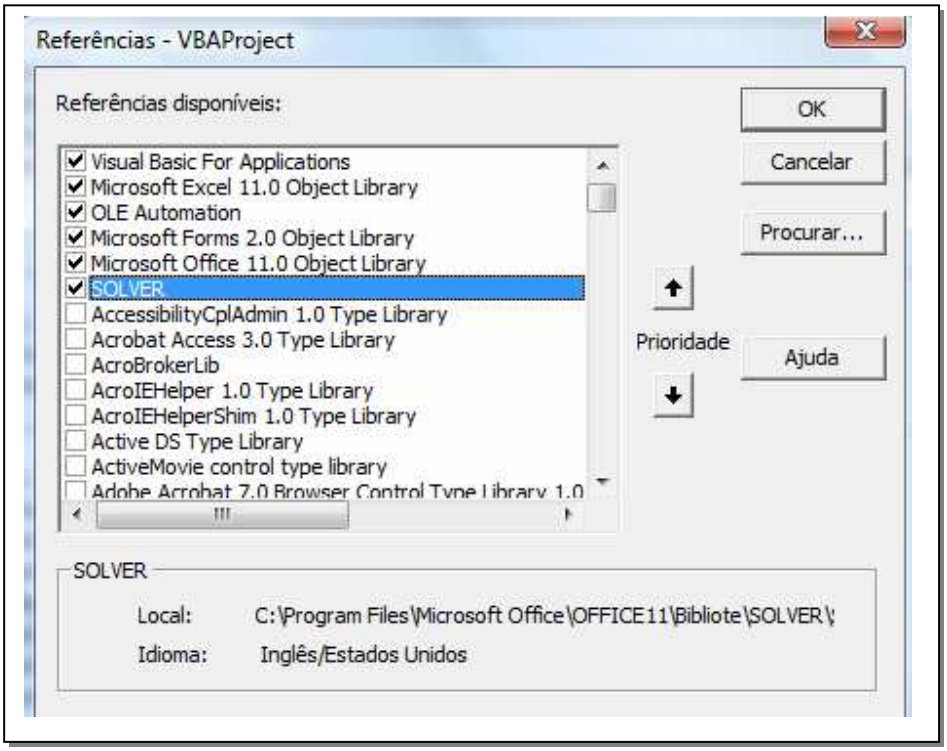



Figura 7.1

Uma referência irá aparecer na janela de Projeto:

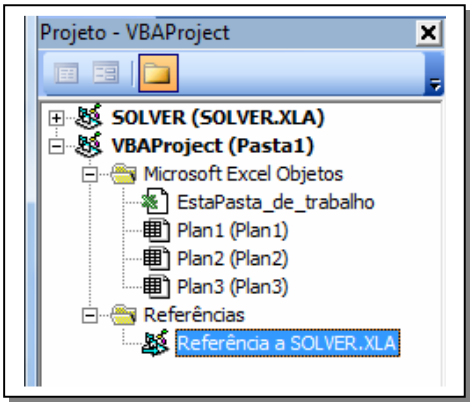



Figura 7.2

Todas as funções do Solver começam com a palavra **Solver**. As mais usadas são SolverReset, SolverOk, SolverAdd, SolverOptions e SolverSolver, que serão explicadas abaixo. Para visualizar a lista completa, entre no Pesquisador de Objeto, selecione a biblioteca Solver, e então, o grupo VBA_Functions.

## Funções:

### -SolverReset

Apaga todas as configurações anteriores para um novo cálculo.

### -SolverOK

Essa função realiza três tarefas:

(1) Identifica a célula que contém a função objetivo
(2) Especifica se o problema é de maximização ou minimização
(3) Identifica as células com as variáveis de decisão.

Exemplo:

```
SolverOK SetCell:=Range("Profit"), MaxMinVal=:1, ByChange:=Range("Quantities")
```

O argumento MaxMinVal, recebe 1 para problemas de maximização e 2 de minimização.

### -SolverAdd

Essa função adiciona uma nova restrição cada vez que é chamada e tem três argumentos.

O argumento central é um índice de relação que tem valores 1 para "**<=**", 2 para "**=**", 3 para "**>=**", 4 para "**inteiro**" e 5 para "**binário**". Os outros argumentos serão as células relacionadas. Para as desigualdades, o primeiro argumento é sempre especificado como um Range, e o terceiro como uma string ou número.

Exemplo:

```
SolverAdd CellRef:=Range('Used"), Relation:=1, FormulaText:=0
```

-**SolverOptions**

Essa função permite escolher qualquer uma das opções da janela de opções do Solver:


Figura 7.3

Pelo fato de todas as opções não serem obrigatórias, basta listar somente os argumentos desejados, separando-os por vírgula. Uma dica é utilizar o Intelicensse digitando SolverOptions e espaço para ver os nomes dos vários argumentos possíveis.

```
SolverOptions
SolverOptions([MaxTime], [Iterations], [Precision], [AssumeLinear], [StepThru], [Estimates], [Derivatives], [SearchOption], [IntTolerance],
[Scaling], [Convergence], [AssumeNonNeg])
```

Figura 7.4

Exemplo:

```
SolverOptions AssumeLinear:=True, AssumeNonneg:=True, Precision:=0.00001
```

**-SolverSolve**

Essa função é equivalente a clicar no botão "Resolver" na usual janela do Solver para executar a otimização.

SolverSolve retorna um valor inteiro, sendo 0 sinalizando que o Solver foi executado com sucesso, 4 para não convergência e 5 para solução não factível.

É conveniente usar o argumento UserFinish:=True, para que a janela de resultados padrão não apareça ao final, evitando interrupções ao longo da execução.

Exemplo:

```
Dim result As Integer
result = SolverSolve(UserFinish:=True)
If result = 5 Then
        MsgBox "Não existe nenhuma solução factível."
End if
```

# 8. Importando dados do Access

Muitas vezes o desenvolvimento de uma modelagem exige o acesso a uma maior quantidade de dados de forma que a entrada destes dados por uma User Form torna-se ineficiente. Estes dados podem estar em um outro arquivo de Excel, ou até em um servidor de banco de dados. Existem várias tecnologias para a importação destes dados, sendo o ADO(ActiveX Data Objects) uma das mais recentes e compatíveis com o VBA para o Excel.

## 8.1 Relational Databases

O Access é um software que lida com "relational Databases", que são banco de dados que tem alguma relação entre si. Esse tipo de banco reduz a duplicação de linhas que existiriam se todas as informações estivessem em um único banco de dados.

A linguagem Structured Query Language (SQL) foi desenvolvida para retornar dados de bancos de dados relacionados. Ele se aplica de certa forma à todos os sistemas de bancos de dados como o Access, SQL Server, Oracle, etc. Porém esta linguagem não é percebida ao trabalhar com o Access, pois este possui uma interface amigável ao usuário, que esconde as sentenças da linguagem.

Para desenvolver a modelagem serão utilizados bancos de dados do Access e portanto, a linguagem SQL dentro do VBA, porém de forma bem simples. Para informações mais detalhadas a referência [3].

## 8.2 ADO

A tecnologia ADO permite ao desenvolvedor escrever códigos relativamente simples para buscar um banco de dados externos. Para utilizar o ADO, é necessário inserir uma referência a ele. No editor VBE, entre em **Ferramentas->Referências**, procure a biblioteca **Microsoft ActiveX Data Objects 2.x** e clique em OK. Essa referência não irá aparecer no Project Explorer como o Solver, pois nesse campo só aparecem items não desenvolvidos pela Microsoft.
Para se referir à um objeto do ADO, use o prefixo **ADODB**, como em ADODB.Connection. Assim que digitar ADODB. e um espaço, uma ajuda sobre a biblioteca será dada pelo Intellisense. Pode-se consultar ajuda também pelo Pesquisador de Objetos.

### Algumas tarefas:

#### Abrir uma conexão com o banco de dados:

Primeiramente, declarar uma variável no formato de um objeto do tipo Connection, especificar aonde o arquivo está, e qual seu provedor, no caso do Access o provedor é Microsof Jet 4.0 OLE DB Provider. Então, abrir o arquivo e fechar a conexão. Segue um exemplo:

```
    Dim cn as New ADOBD.Connection
'Esse código procura o arquivo Arquivo.mdb na pasta de trabalho atual em que o arquivo de
    'Excel se encontra.
With cn
        .ConnectionString="Data Source" & ThisWorkbook.Path & "\Arquivo.mdb"
        .Provider = "Microsof Jet 4.0 OLE DB Provider"
        .Open
End With

cn.Close
```

## Abrir um Registro

Um registro é um banco de dados temporário que fica armazenado na memória. Para acessá-lo, declara-se uma variável no formato de um objeto do tipo Recordset, depois utilizando uma variável do tipo string que contém declarações em SQL, abrimos o registro com o método Open. Os dois primeiros argumentos deste método são uma SQL string e o objeto de conexão que já foi aberto. Segue o código do procedimento:

```
rs as New ADOBD.Recordset
'Abrir o registro
rs.Open SQL, cn
´Fecha o registro
rs.Close
```

O ADO pode ser usado para abrir outros tipos de bancos de dados sem ser no formato Access, como por exemplo, bancos de dados armazenados em servidores.

A única diferença nos códigos acima para abrir estes bancos são nas propriedades ConnectionString e Provider.

## Procurar alguma informação no banco de dados:

```
    'Este código executa as declarações em cada linha do registro até
encontrar o fim do arquivo, (EOF significa end of file).
With rs
        Do Until .EOF
    Delacrações
    .MoveNext
    Loop
    End With
```

**Alguns comandos úteis em SQL:**

**SELECT:** lista os campos que contém os dados desejados.
**FROM:** especifica a tabela ou tabelas que contém estes dados.
**WHERE:** Lista um critério específico.
**GROUP BY:** permite o agrupamentos de alguns dados, por exemplo o total de carros de certa marca. Neste caso o comando seria GROUP BY marca.
**ORDER BY:** Permite ordenar o banco desejado.

# 9. Implementando a Modelagem

## 9.1 Pré-Modelagem

Antes de iniciar uma modelagem em VBA, é útil que o programador pense nos seguintes tópicos:

**1**-Como os dados serão inseridos
**2**-Como criar o modelo com os dados de entrada e obter a resolução do problema
**3**-Como os resultados serão exibidos

Para solucionar o segundo tópico é necessário conhecimento matemático de modelagem, e isso não será tratado neste tutorial. Para mais informações sobre a modelagem a ser desenvolvida consulte a referência [2].

Como resposta do primeiro tópico, o programador pode escolher a entrada de dados por caixas de diálogo, construindo User Forms. Mas como na maioria das vezes existe uma grande quantidade de dados a ser trabalhada, essa opção torna-se inviável. Muitas vezes os dados estão em outros arquivos do Excel, ou em bancos de dados externos, como no Access. O modelo que segue tratará desta última opção.

O terceiro tópico tem como opção basicamente o uso de gráfico ou tabelas, e é preferível decidir de antemão quais dados serão analisados e quais tipos de gráficos serão reportados.

### Observações Importantes

**1.** É importante deixar claro ao usuário o que a aplicação faz e suas limitações. Uma forma de se fazer isso é fazer uma criando uma planilha explicativa com a descrição do modelo a ser desenvolvido, que o usuário irá ver assim que abrir o arquivo em Excel. As explicações mais detalhadas podem ser dadas em caixas de diálogo, por exemplo.

**2.** Tenha o hábito de criar uma sub-rotina principal que chama as outras sub-rotinas. É preferível que estas executem pequenas tarefas ao invés de se usar uma única sub-rotina que realize muitas tarefas.

**3.** Decida o que pode ser feito graficamente em vez de em tempo real. A interface do Excel oferece maneiras muito rápidas de criar gráficos ou tabelas, ou inserir fórmulas. Portanto se no modelo, por exemplo, será exibido um gráfico, pode-se criar uma planilha com o "template" desse gráfico e então manipulá-lo pelo VBA, alterando o que for necessário. A idéia é que quanto mais elementos puderem ser feitos graficamente pela interface do Excel, menos esforço será despendido na programação.

## 9.2 Execução da Aplicação

A aplicação está no arquivo Transporte.xlsm. Quando esse arquivo é aberto, a planilha "Explicativa" da figura 9.1 aparece. Quando o botão (Executar a aplicação) é clicado, é apresentado ao usuário uma caixa de diálogo como o da figura 9.2, contendo uma lista com todas as fábricas e revendedores disponíveis no banco de dados. O usuário pode selecionar quantos locais desejar e os dados destes locais são importados do banco de dados para a planilha "Modelo" (figura 9.3). Então, o modelo em si é desenvolvido através da chamada do Solver, e os resultados são transferidos para a planilha "Resultados" (figura 9.4). O usuário pode então, executar novamente a aplicação selecionando diferentes fábricas e revendedores.

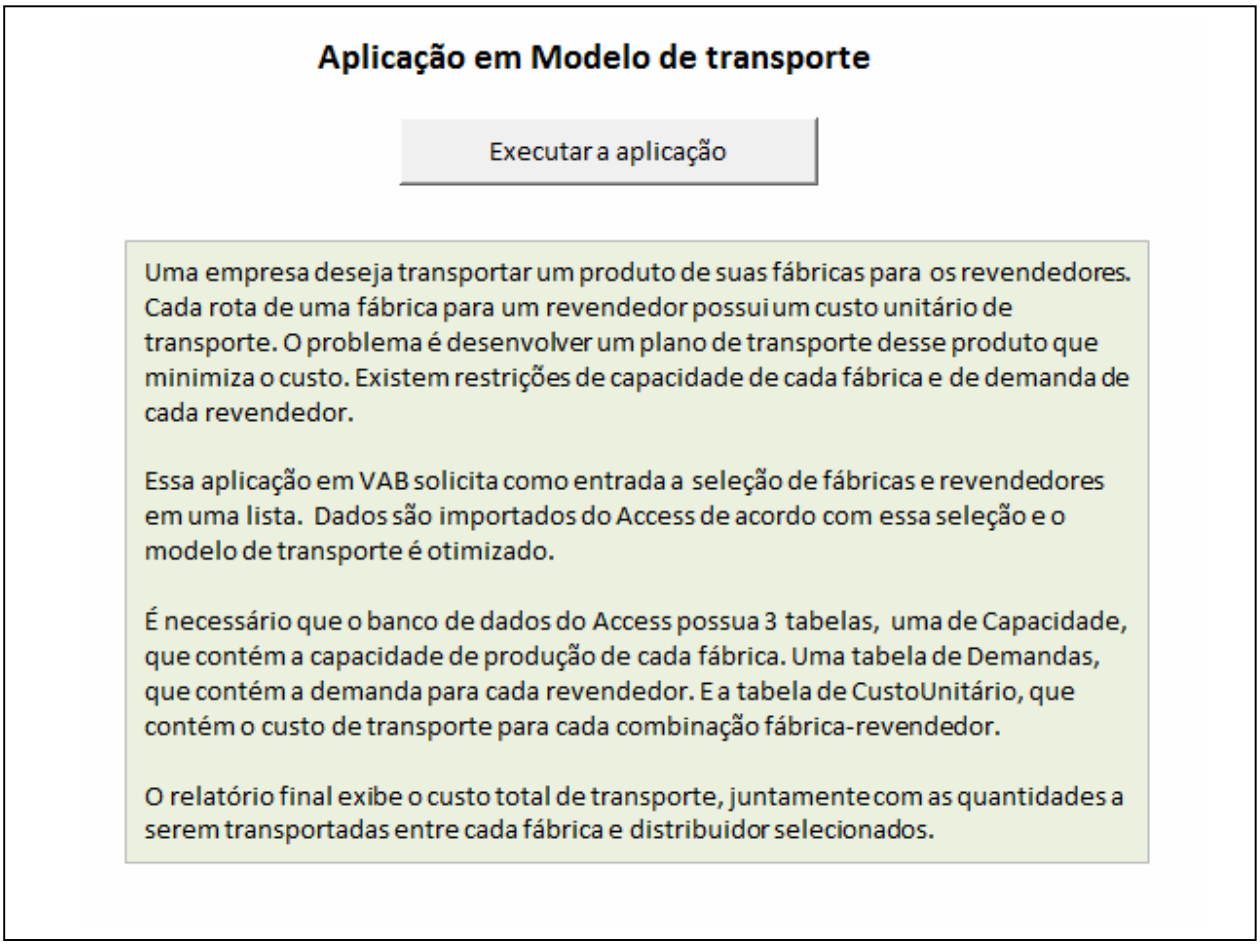



Figura 9.1


Figura 9.2

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problema de Transporte | | | | | | | | | | |
| | | | | | | | | | | |
| | Custo Unitário | | Revendedores | | | | | | | |
| | | | Minneap | Chicago | Milwauke | Detroit | Cleveland | Toronto | Baltimore | Capacidades |
| | Fábricas | Seattle | $20.04 | $23.34 | $14.30 | $27.14 | $14.38 | $19.76 | $21.31 | 1300 |
| | | Oakland | $22.37 | $16.67 | $20.37 | $22.42 | $24.09 | $12.23 | $19.99 | 1200 |
| | | Los Angeles | $13.71 | $25.45 | $15.87 | $31.54 | $26.49 | $27.70 | $17.04 | 1000 |
| | | San Diego | $21.35 | $26.41 | $18.86 | $20.58 | $21.70 | $10.68 | $21.16 | 1500 |
| | | | | | | | | | | |
| | | Demandas | 500 | 500 | 400 | 450 | 200 | 500 | 400 | |
| | | | | | | | | | | |
| | Remessas | | | | | | | | | Quantidades enviadas |
| | | | 0 | 0 | 400 | 0 | 200 | 0 | 0 | 600 |
| | | | 0 | 500 | 0 | 0 | 0 | 0 | 0 | 500 |
| | | | 500 | 0 | 0 | 0 | 0 | 0 | 400 | 900 |
| | | | 0 | 0 | 0 | 450 | 0 | 500 | 0 | 950 |
| | | Quantidades recebidas | 500 | 500 | 400 | 450 | 200 | 500 | 400 | |
| | | | | | | | | | | |
| | Custo Total | | $45,203 | | | | | | | |

Figura 9.3

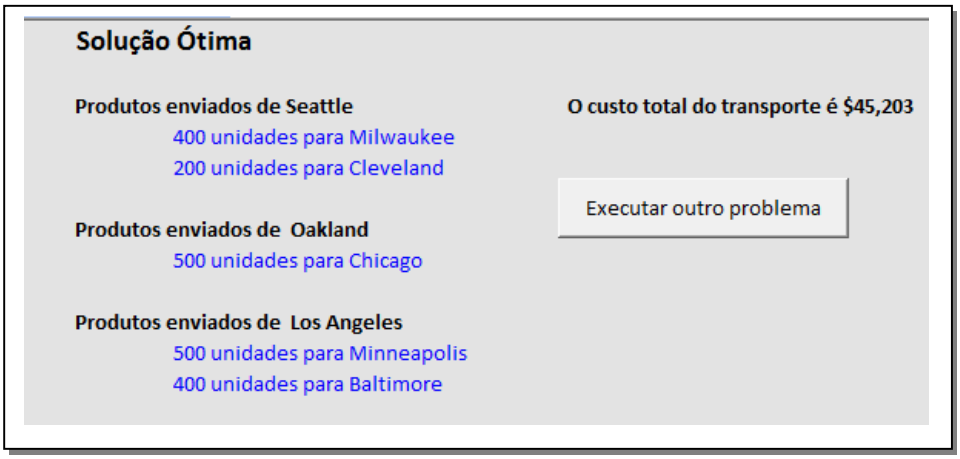
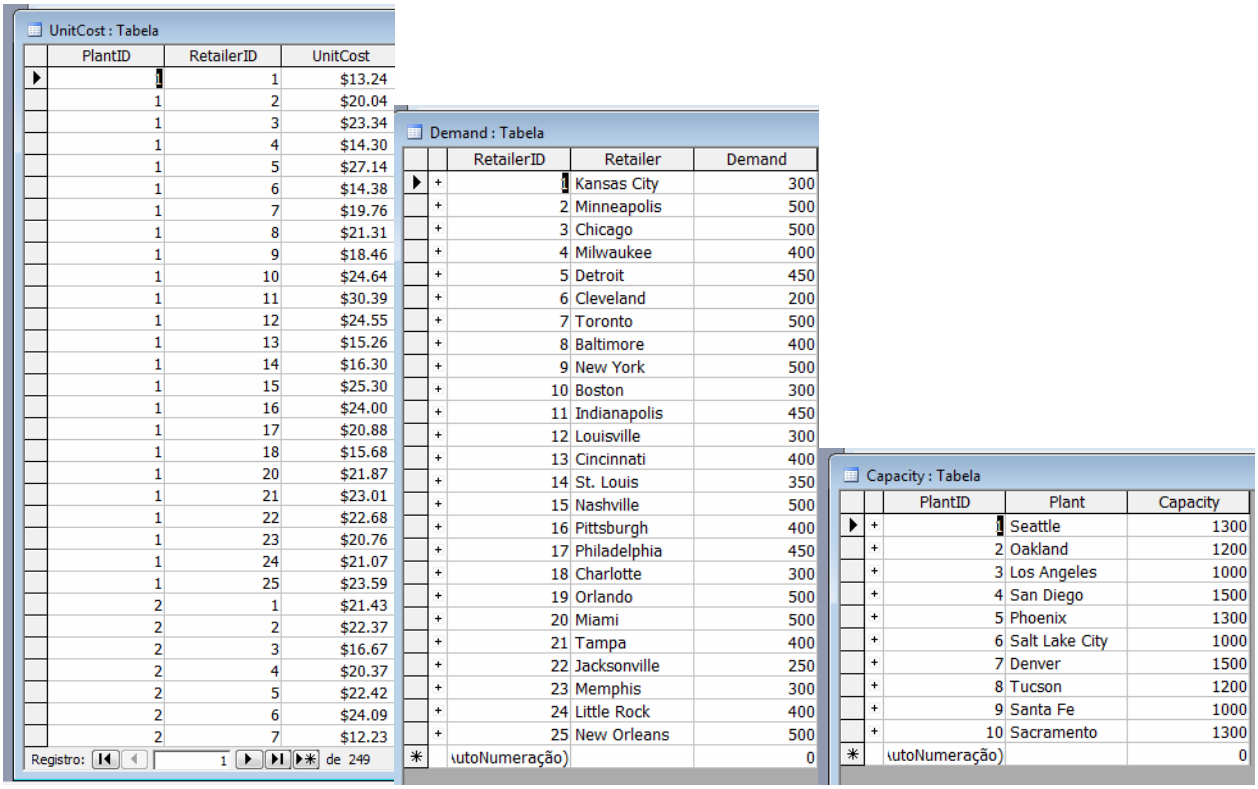



Figura 9.4

Para que a aplicação funcione corretamente, é necessário que o arquivo em Access se chame Transporte.mdb e esteja na mesma pasta que o arquivo Transporte.xlsm. Este arquivo precisa conter 3 tabelas (Demand, Capacity e UnitCost) com o formato como mostra a figura abaixo:



UnitCost : Tabela

| PlantID | RetailerID | UnitCost |
|---|---|---|
| 1 | 1 | $13.24 |
| 1 | 2 | $20.04 |
| 1 | 3 | $23.34 |
| 1 | 4 | $14.30 |
| 1 | 5 | $27.14 |
| 1 | 6 | $14.38 |
| 1 | 7 | $19.76 |
| 1 | 8 | $21.31 |
| 1 | 9 | $18.46 |
| 1 | 10 | $24.64 |
| 1 | 11 | $30.39 |
| 1 | 12 | $24.55 |
| 1 | 13 | $15.26 |
| 1 | 14 | $16.30 |
| 1 | 15 | $25.30 |
| 1 | 16 | $24.00 |
| 1 | 17 | $20.88 |
| 1 | 18 | $15.68 |
| 1 | 20 | $21.87 |
| 1 | 21 | $23.01 |
| 1 | 22 | $22.68 |
| 1 | 23 | $20.76 |
| 1 | 24 | $21.07 |
| 1 | 25 | $23.59 |
| 2 | 1 | $21.43 |
| 2 | 2 | $22.37 |
| 2 | 3 | $16.67 |
| 2 | 4 | $20.37 |
| 2 | 5 | $22.42 |
| 2 | 6 | $24.09 |
| 2 | 7 | $12.23 |

Registro: 1 de 249

Demand : Tabela

| RetailerID | Retailer | Demand |
|---|---|---|
| 1 | Kansas City | 300 |
| 2 | Minneapolis | 500 |
| 3 | Chicago | 500 |
| 4 | Milwaukee | 400 |
| 5 | Detroit | 450 |
| 6 | Cleveland | 200 |
| 7 | Toronto | 500 |
| 8 | Baltimore | 400 |
| 9 | New York | 500 |
| 10 | Boston | 300 |
| 11 | Indianapolis | 450 |
| 12 | Louisville | 300 |
| 13 | Cincinnati | 400 |
| 14 | St. Louis | 350 |
| 15 | Nashville | 500 |
| 16 | Pittsburgh | 400 |
| 17 | Philadelphia | 450 |
| 18 | Charlotte | 300 |
| 19 | Orlando | 500 |
| 20 | Miami | 500 |
| 21 | Tampa | 400 |
| 22 | Jacksonville | 250 |
| 23 | Memphis | 300 |
| 24 | Little Rock | 400 |
| 25 | New Orleans | 500 |
| (AutoNumeração) | | 0 |

Capacity : Tabela

| PlantID | Plant | Capacity |
|---|---|---|
| 1 | Seattle | 1300 |
| 2 | Oakland | 1200 |
| 3 | Los Angeles | 1000 |
| 4 | San Diego | 1500 |
| 5 | Phoenix | 1300 |
| 6 | Salt Lake City | 1000 |
| 7 | Denver | 1500 |
| 8 | Tucson | 1200 |
| 9 | Santa Fe | 1000 |
| 10 | Sacramento | 1300 |
| (AutoNumeração) | | 0 |

Figura 9.5

Esse arquivo pode ser alterado desde que mantenha o formato acima. Pode-se adicionar ou excluir mais fábricas e revendedores ou alterar nomes e custos.

# 10. Programação em VBA

Esse capítulo mostra todas as macros utilizadas na construção do modelo. Algumas são bem simples e outras mais complexas, porém com as informações que foram dadas até aqui sobre a linguagem e com os comentários que seguem dentro de cada sub-rotina, é possível compreendê-las.

## 10.1 Configurações iniciais

Como o modelo utilizará o **Solver** e o **ADO**, é necessário inserir uma referência a eles como foi explicado nos capítulos anteriores. Vale lembrar que a referência ao

ADO não irá aparecer no Project Explorer. Também cria-se um **módulo** para a inserção das sub-rotinas.

## 10.2 Criando o máximo no modo gráfico

Primeiramente, cria-se o máximo do layout no modo gráfico. Tudo o que não for variar ao longo da execução do modelo pode ser feito nesse modo. No modelo, criam-se 3 planilhas. A "**Explicativa**" pode ser criada totalmente no modo gráfico como estava na figura 9.1. Nesta planilha é inserido um botão ligado à sub-rotina MainTransport. Como a planilha "**Modelo**" será preenchida praticamente em tempo real, pois o problema tem tamanho variável, o que pode ser feito em modo gráfico está na figura 10.1. E o que pode ser feito da planilha "**Resultados**" está na figura 10.2. Nesta planilha também existe um botão que chama a sub-rotina MainTrasport.

Ao criar as planilhas, automaticamente uma referência à elas irá aparecer no Project Explorer do VBE.

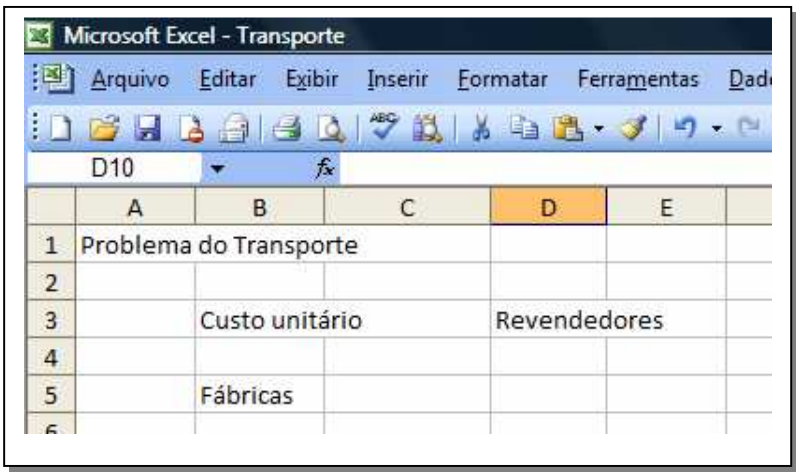



Figura 10.1

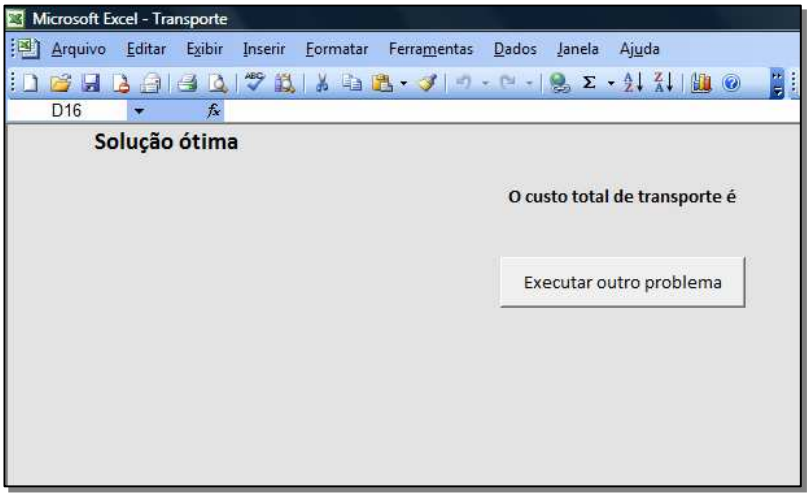



Figura 10.2

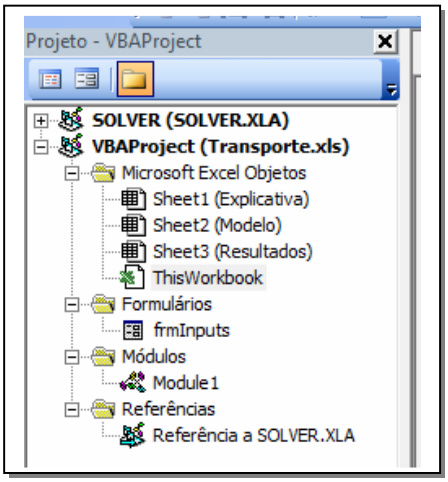



Figura 10.3

## 10.3 Janela ThisWorkbook

Para garantir que a planilha explicativa apareça quando o arquivo é aberto, e que as outras não apareçam, o seguinte código é implementado na janela ThisWorkbook.

```
Private Sub Workbook_Open()
    Worksheets("Modelo").Visible = False
    Worksheets("Resultados").Visible = False
    Worksheets("Explicativa").Activate
    Range("G18").Select
End Sub
```

## 10.4 Inserindo Formulários e EventHandlers

Cria-se um formulário (frmInputs) como foi descrito no capítulo 6, com um layout contendo 2 caixas de listagem nomeadas lbPlants e lbRetailers e dois botões: Ok e Cancel. Veja a figura abaixo:

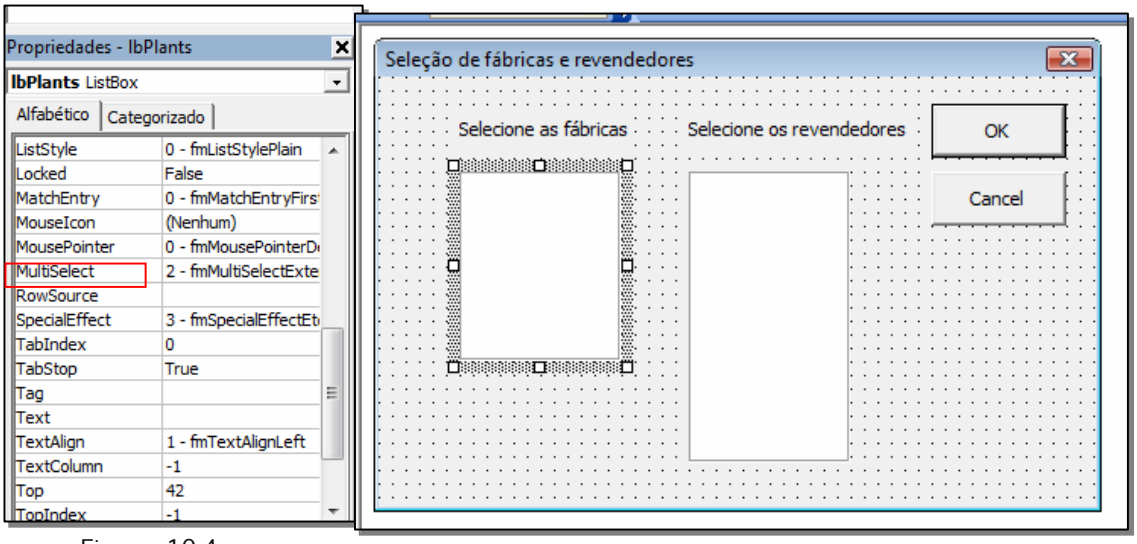



Figura 10.4

Cada caixa possui a propriedade de múltipla seleção (MultiSelect) escolhida como 2-fmMultiSelectExtended, que permite o usuário selecionar mais de um item das caixas segurando a tecla Ctrl.

### EventHandlers

A sub-rotina btnCancel_Click executa sua função usual, saindo do formulário e terminando a aplicação. A subrotina btnOK_click, usa a propriedade Selected que é "0-based indexing" para capturar as fábricas e revendedores selecionados em vetores públicos "1-based indexing" (selectedPlants e selectedRetailers) que estão declarados no início do módulo. Depois de capturar esses vetores, constrói-se com eles duas strings PlantsList e RetailersList que serão usados para capturar os custos unitários. Dessa forma se o usuário selecionar, por exemplo, as fábricas Boston, New York, Toronto, PlantsList será "('Boston','New York,'Toronto')".

A sub-rotina UserForm_Initialize preenche as caixas de listagem com todas as fábricas e revendedores existentes no banco de dados, que estão armazenados nos vetores existingPlant e existingRetailer.

O código completo encontra-se abaixo:

```
Private Sub btnCancel_Click()
Unload Me
End
End Sub

Private Sub btnOK_Click()
' Preenche um vetor, selectedPlant, com as fábricas selecionadas.
nSelectedPlants = 0
For i = 1 To lbPlants.ListCount
If lbPlants.Selected(i - 1) Then
nSelectedPlants = nSelectedPlants + 1
ReDim Preserve selectedPlant(nSelectedPlants)
selectedPlant(nSelectedPlants) = lbPlants.List(i - 1)
End If
Next
If nSelectedPlants = 0 Then
MsgBox "Favor selecionar ao menos uma fábrica.", vbExclamation, _
"Seleção requerida"
Exit Sub
End If

' Constrói uma string, PlantList, que tem a seguinte estrutura:
' ('_','_','_'), dependendo das fábricas selecionadas.
plantList = "("
For i = 1 To nSelectedPlants
If i < nSelectedPlants Then
plantList = plantList & "'" & selectedPlant(i) & "',"
Else
plantList = plantList & "'" & selectedPlant(i) & "')"
End If
Next

' Preenche um vetor, selectedRev, com os revendedores selecionados.
nSelectedRetailers = 0
For i = 1 To lbRetailers.ListCount
If lbRetailers.Selected(i - 1) Then
nSelectedRetailers = nSelectedRetailers + 1
ReDim Preserve selectedRetailer(nSelectedRetailers)
selectedRetailer(nSelectedRetailers) = lbRetailers.List(i - 1)
End If
Next
If nSelectedRetailers = 0 Then
MsgBox "Favor selecionar ao menos um revendedor.", vbExclamation, _
"Seleção requerida"
Exit Sub
End If
```

```
selectedRetailer(nSelectedRetailers) = lbRetailers.List(i - 1)
End If
Next
If nSelectedRetailers = 0 Then
MsgBox "Favor selecionar ao menos um revendedor.", vbExclamation, _
"Seleção requerida"
Exit Sub
End If

' Constrói uma string, RevList, que tem a seguinte estrutura:
' ('_','_','_'), dependendo dos revendedores selecionados.
retailerList = "("
For i = 1 To nSelectedRetailers
If i < nSelectedRetailers Then
retailerList = retailerList & "'" & selectedRetailer(i) & "',"
Else
retailerList = retailerList & "'" & selectedRetailer(i) & "')"
End If
Next

Unload Me
End Sub

Private Sub UserForm_Initialize()
' Preenche as caixas de listagem com os vetores.
lbPlants.List = existingPlant
lbRetailers.List = existingRetailer
End Sub
```

## 10.5 Construção de Subrotinas

Dentro da janela “Module 1”, antes de iniciar qualquer subrotina, definimos a opção de indexação como “1-base indexing” e forçamos a declaração de variáveis antes dela ser usada.

Também declaramos as variáveis públicas e as que são visíveis em todo módulo:

```
Option Explicit
Option Base 1

Public existingPlant() As String, existingRetailer() As String
Public nSelectedPlants As Integer, nSelectedRetailers As Integer
Public selectedPlant() As String, selectedRetailer() As String
Public plantList As String, retailerList As String

' Declara objetos de conexão e registro do ADO.
Dim cn As New ADODB.Connection, rs As New ADODB.Recordset
```

## Macro "Principal"

```
Sub Main Transport()
    ' Essa é a macro que é executada quando o botão anexado na planilha
    ' "Explicativa" é clicado.

    ' Desliga a função de atualização da tela do Excel para aumentar o desempenho da
    ' aplicação
    Application.ScreenUpdating = False

    ' Abre uma conexão com o banco de dados do Access, assumindo que esse
    ' banco e a pasta de trabalho do Excel estejam na mesma pasta.
    With cn
        .ConnectionString = "Data Source=" & ThisWorkbook.Path & "\Transporte.mdb"
        .Provider = "Microsoft Jet 4.0 OLE DB Provider"
        .Open
    End With

    ' Importa os dados desejados do banco.
    Call GetPlantAndRetailers
    frmInputs.Show
    Call CheckSolverLimit
    Call EnterModelData

    ' Fecha a conexão com o banco de dados.
    cn.Close

    ' Configura e soluciona o modelo.
    Call EnterFormulas
    Call SetAndRunSolver
End Sub
```

Quando o botão “Executar a aplicação” na tabela explicativa é clicado, a seguinte macro é rodada. Essa macro chama outras sub-rotinas que extraem os dados do banco de dados do Access, desenvolvem o modelo, executam o Solver e informam o resultado.

## Macro GetPlantAndRetailers

Essa macro importa os dados das tabela “Capacity” e “Demand”. Isso pode ser feito através de uma sentença em SQL: “Select Plant From Capacity”. Então, os nomes das fábricas são armazenados em vetores. O mesmo ocorre com os nomes dos revendedores.

```
Sub GetPlantAndRetailers()
' Essa subrotina pega os nomes das fábricas e revendedores no banco de dados
' para posteriormente preencher as caixas de listagem

Dim nExisitingPlants As Integer, nExisitingRetailers As Integer
Dim i As Integer, j As Integer
Dim SQL As String

' Pega os nomes das fábricas na tabela “Capacity” para preencher o vetor existingPlant.
SQL = "Select Plant From Capacity"
With rs
.Open SQL, cn
nExisitingPlants = 0
Do While Not .EOF
nExisitingPlants = nExisitingPlants + 1
ReDim Preserve existingPlant(nExisitingPlants)
existingPlant(nExisitingPlants) = .Fields("Plant").Value
.MoveNext
Loop
.Close
End With

' Pega os nomes dos revendedores na tabela “Demand” para preencher
' o vetor existingRetailers.
SQL = "Select Retailer from Demand"
With rs
.Open SQL, cn
nExisitingRetailers = 0
Do While Not .EOF
nExisitingRetailers = nExisitingRetailers + 1
```

```
ReDim Preserve existingRetailer(nExisitingRetailers)
existingRetailer(nExisitingRetailers) = .Fields("Retailer").Value
.MoveNext
Loop
.Close
End With


End Sub
```

## Macro CheckSolverLimit

Essa macro checa se o limite de variáveis de decisão do SOLVER foi excedido.

```
Sub CheckSolverLimit()
Dim nChangingCells As Integer
nChangingCells = nSelectedPlants * nSelectedRetailers
If nChangingCells > 200 Then
MsgBox "De acordo com suas seleções, existem " & nChangingCells _
& " células variáveis no modelo (" & nSelectedPlants & " x " _
& nSelectedRetailers & "). O máximo que o Solver suporta são 200. " _
& "Escolha menos fábricas or revendedores.", _
vbExclamation, "Modelo muito grande"
frmInputs.Show
End If
End Sub
```

## Macro EnterModelData

Essa macro importa as demandas, capacidades e custos unitários correspondentes às fábricas e revendedores selecionados. Então coloca-os nas células apropriadas da planilha “Modelo”.

```
Sub EnterModelData()
' A seguinte macro utiliza o ADO para trasnportar os dados
' do banco de dados para as células da planilha “Modelo”.
Dim SQL As String
Dim i As Integer, j As Integer
Dim topCell As Range

' Apaga tudo da plhanilha “Modelo”.
With Worksheets("Modelo")
.Cells.Clear
.Activate
End With

'Nomeia as células.
Range("A1").Value = "Problema de Transporte"
Range("B3").Value = "Custos Unitários"
Range("B5").Value = "Fábricas"
Range("D3").Value = "Revendedores"

Set topCell = Range("C4")
' Adiciona os nomes das fábricas e revendedores.
With topCell
For i = 1 To nSelectedPlants
.Offset(i, 0).Value = selectedPlant(i)
Next
For j = 1 To nSelectedRetailers
.Offset(0, j).Value = selectedRetailer(j)
Next
End With

' Pega os dados com a seguinte sentença em SQL, e preenche as células de
'Custo Unitário.
SQL = "Select UC.UnitCost " _
& "From (UnitCost UC Inner Join Capacity C On UC.PlantID = C.PlantID) " _
& "Inner Join Demand D On UC.RetailerID = D.RetailerID " _
& "Where C.Plant In " & plantList & " And D.Retailer In " & retailerList
With rs
.Open SQL, cn
For i = 1 To nSelectedPlants
For j = 1 To nSelectedRetailers
topCell.Offset(i, j).Value = .Fields("UnitCost").Value
.MoveNext
Next
Next
.Close
End With
```

```
.Open SQL, cn
For i = 1 To nSelectedPlants
For j = 1 To nSelectedRetailers
topCell.Offset(i, j).Value = .Fields("UnitCost").Value
.MoveNext
Next
Next
.Close
End With

' Nomeia as células de Custo Unitário
With topCell
Range(.Offset(1, 1), .Offset(nSelectedPlants, nSelectedRetailers)) _
.Name = "UnitCosts"
End With

' Preenche os valores de capacidade.
Set topCell = Range("C4").Offset(0, nSelectedRetailers + 2)
topCell.Value = "Capacitidades"
SQL = "Select Capacity From Capacity " & _
"Where Plant In " & plantList
With rs
.Open SQL, cn
For i = 1 To nSelectedPlants
topCell.Offset(i, 0).Value = .Fields("Capacity").Value
.MoveNext
Next
.Close
End With

' Nomeia as células de capacidades.
With topCell
Range(.Offset(1, 0), .Offset(nSelectedPlants, 0)).Name = "Capacities"
End With

' Preenche os valores de demanda.
Set topCell = Range("C4").Offset(nSelectedPlants + 2, 0)
topCell.Value = "Demandas"
SQL = "Select Demand From Demand " & _
"Where Retailer In " & retailerList
With rs
.Open SQL, cn
For j = 1 To nSelectedRetailers
topCell.Offset(0, j).Value = .Fields("Demand").Value
.MoveNext
Next
.Close
End With
```

```
topCell.Value = "Demandas"

SQL = "Select Demand From Demand " & _
"Where Retailer In " & retailerList
With rs
.Open SQL, cn
For j = 1 To nSelectedRetailers
topCell.Offset(0, j).Value = .Fields("Demand").Value
.MoveNext
Next
.Close
End With

' Nomeia as células de demanda.
With topCell
Range(.Offset(0, 1), .Offset(0, nSelectedRetailers)).Name = "Demands"
End With

End Sub
```

Ao final dessa macro, todos os dados necessários para o problema estão devidamente dispostos na planilha “Modelo.

## Macro EnterFormulas

Essa macro realiza os cálculos necessários para a construção das restrições do problema.

```
Sub EnterFormulas()

Dim outOfRange As Range, intoRange As Range
Dim topCell As Range

Worksheets("Model").Activate

' Seleciona as células das remessas.
Range("B4").Offset(nSelectedPlants + 4, 0).Value = "Remessas"

Set topCell = Range("C4").Offset(nSelectedPlants + 4, 0)
With Range(topCell.Offset(1, 1), _
topCell.Offset(nSelectedPlants, nSelectedRetailers))
.Name = "Shipments"
.Value = 0
```

```
.BorderAround Weight:=xlMedium, ColorIndex:=3
End With

' Entra com as formulas para as somas das linhas e colunas das
‘"Quantidades enviadas" e  "Quantidade recebidas".
Set topCell = Range("Shipments").Cells(1)
With topCell
.Offset(-1, nSelectedRetailers).Value = "Quantidades enviadas"
.Offset(nSelectedPlants, -1).Value = "Quantidades recebidas"
Set outOfRange = Range(.Offset(0, nSelectedRetailers), _
.Offset(nSelectedPlants - 1, nSelectedRetailers))
Set intoRange = Range(.Offset(nSelectedPlants, 0), _
.Offset(nSelectedPlants, nSelectedRetailers - 1))
End With
With outOfRange
.Name = "SentOut"
.FormulaR1C1 = "=SUM(RC[-" & nSelectedRetailers & "]:RC[-1])"
End With
With intoRange
.Name = "SentIn"
.FormulaR1C1 = "=SUM(R[-" & nSelectedPlants & "]C:R[-1]C)"
End With
End With
' Calcula a célula de custo total.
Set topCell = Range("SentIn").Item(1).Offset(2, 0)
With topCell
.Formula = "=SumProduct(UnitCosts,Shipments)"
.Name = "TotalCost"
.NumberFormat = "R$#,##0_);(R$#,##0)"
.Offset(0, -2).Value = "Custo Total"

Range("A1").Select
End Sub
```

## Macro SetupAndRunSolver()

Essa macro configura e executa o Solver. Também checa se alguma solução é infactível.

```
Sub SetupAndRunSolver()
Dim solverStatus As Integer

With Worksheets("Model")
.Visible = True
.Activate
End With

SolverReset
SolverAdd Range("Shipments"), 3, 0
SolverAdd Range("SentOut"), 1, "Capacities"
SolverAdd Range("SentIn"), 3, "Demands"
SolverOptions AssumeLinear:=True, AssumeNonNeg:=True
SolverOk SetCell:=Range("TotalCost"), MaxMinVal:=2, _
ByChange:=Range("Shipments")

' Executa o Solver. Se não existir nenhuma solução factível,
' o problema retorna ao início pedindo para o usuário selecionar outras entradas.
solverStatus = SolverSolve(UserFinish:=True)
If solverStatus = 5 Then
MsgBox "Não existe solução factível. Tente uma combinação diferente de " _
& "fábricas e revendedores.", vbExclamation, "Infactibilidade"
Call MainTransport
Else
Worksheets("Model").Visible = False
Call CreateReport
End If
End Sub
```

## Macro CreateReport

Transfere o custo total e as informações sobre as rotas com remessas positivas para a planilha “Resultados”. Também será informado se não existir nenhum transporte a partir de uma determinada fábrica.

```
Sub CreateReport()
Dim i As Integer, j As Integer
Dim nShippedTo As Integer, amountShipped As Integer
Dim topCell As Range

' Apaga todo o conteúdo da planilha menos o botão para executar outro problema.
With Worksheets("Resultados")
.Cells.Clear
.Visible = True
.Activate
End With

' Entra com os valores da células e configurações de cor e estilo.
With Range("B1")
.Value = "Solução ótima"
With .Font
.Size = 14
.Bold = True
End With
End With
With ActiveWindow
.DisplayGridlines = False
.DisplayHeadings = False
End With
Cells.Interior.ColorIndex = 40
Columns("B:B").Font.ColorIndex = 1
Columns("C:C").Font.ColorIndex = 5
Range("B1").Font.ColorIndex = 1

' Para cada rota com quantidades positivas, imprime quanto é transportado.
Set topCell = Range("B3")
For i = 1 To nSelectedPlants
If Range("SentOut").Cells(i).Value > 0.1 Then
With topCell
.Value = " Produtos enviados de " & selectedPlant(i)
.Font.Bold = True
End With
nShippedTo = 0
For j = 1 To nSelectedRetailers
amountShipped = Range("Shipments").Cells(i, j).Value
If amountShipped > 0.01 Then
nShippedTo = nShippedTo + 1
topCell.Offset(nShippedTo, 1).Value = _
amountShipped & " unidades para " & selectedRetailer(j)
End If
Next
Set topCell = topCell.Offset(nShippedTo + 2, 0)
Else
```

```
With topCell
.Value = "Nenhum produto enviado de " & selectedPlant(i)
.Font.Bold = True
End With
Set topCell = topCell.Offset(2, 0)
End If
Next

' Imprime o custo total
With Range("G3")
.Value = "O custo total de trasponte é " & _
Format(Range("TotalCost").Value, "R$#,##0;(R$#,##0)")
.Font.Bold = True
End With
Range("A1").Select
End Sub
```

# Conclusão

Pode-se notar pontos favoráveis e desfavoráveis ao implementar uma modelagem utilizando VBA.

A melhor vantagem em utilizar essa linguagem é que ela possui uma ilimitada flexibilidade. Ou seja, com o Excel podemos realizar os cálculos, importar e exportar dados com outras extensões, incluir bibliotecas que fazem uso de outras linguagens, criar visualizações convenientes para o usuário, etc. Ao final, todo o conteúdo está em um único arquivo que pode ser gravado e executado em qualquer outra máquina que possua o software. Além disso, provavelmente todas as empresas possuem o Microsoft Excel instalado em suas máquinas, e a muitas pessoas estão de alguma maneira familiarizadas com sua interface.

A linguagem VBA é uma linguagem fácil de se aprender, visto a grande quantidade de livros e websites sobre o assunto. Além do mais, o pesquisador de objetos e o intellisense são ferramentas que facilitam ainda mais o aprendizado.

Como desvantagem, temos a lenta performance. A execução de uma sub-rotina é feita linha por linha. Cada linha é lida, checada para ver se não há erros na sintaxe, compilada e executada. Isso torna a execução muito lenta dependendo do tamanho do problema, se comparada com outras aplicações desenvolvidas em C, por exemplo.

Além disso, existe a limitação do número de células, cerca de 65.536 linhas para as versões 2003 ou anteriores.

Deve-se também ter cuidado ao manipular o arquivo, pois algumas alterações nas planilhas podem modificar o código, gerando conflitos.

Conclui-se que deve-se ter bom senso ao escolher o VBA como ferramenta para o desenvolvimento de modelos de otimização. Apesar da grande funcionalidade, essa é uma opção viável apenas para problemas de pequena e média escala, e em que minimizar o tempo de execução não seja o principal objetivo.

# Referências

1) Tutorial sobre modelagem utilizando o Solver:
http://www.ime.unicamp.br/~moretti/ms428/2sem2010/TutorialExcel.doc

2) Linear programming and network flows / Mokhtar S. Bazaraa, John J. Jarvis, Hanif D. Sherali.

3) RAMALHO, Jose Antonio A. (Jose Antonio Alves). SQL: a linguagem dos bancos de dados. São Paulo, SP: Berkeley.

# Bibliografia

1) Albright, S. C.; VBA for Modelers Developing Decision Support Systems with Microsoft Office Excel. 2010 ed. South-Western Cengage Learning.