通过把耗时长的函数用c语言实现，并编译成mex函数可以加快执行速度。Matlab本身是不带c语言的编译器的，所以要求你的机器上已经安装有VC,BC或Watcom C中的一种。如果你在安装Matlab时已经设置过编译器，那么现在你应该就可以使用mex命令来编译c语言的程序了。如果当时没有选，就在Matlab里键入mex -setup，下面只要根据提示一步步设置就可以了。【需要注意的是，较低版本的在设置编译器路径时，只能使用路径名称的8字符形式。比如我用的VC装在路径C:\PROGRAM FILES\DEVSTUDIO下，那在设置路径时就要写成：“C:\PROGRA~1”这样设置完之后，mex就可以执行了。】

为了测试你的路径设置正确与否，把下面的程序存为hello.c。  
/\*hello.c\*/  
#include "mex.h"   
void mexFunction(int nlhs, mxArray \*plhs[], int nrhs, const mxArray \*prhs[])   
{ mexPrintf("hello,world!\n");   
}   
假设你把hello.c放在了C:\TEST\下，在Matlab里用CD C:\TEST\ 将当前目录改为C:\ TEST\（注意，仅将C:\TEST\加入搜索路径是没有用的）。

现在敲：  
mex hello.c   
如果一切顺利，编译应该在出现编译器提示信息后正常退出。

如果你已将C:\TEST\加  
入了搜索路径，现在键入hello，程序会在屏幕上打出一行：  
hello,world!

看看C\TEST\目录下，你会发现多了一个文件：HELLO.DLL。这样，第一个mex函数就算完成了。

分析hello.c，可以看到程序的结构是十分简单的，整个程序由一个接口子过程 mexFunction构成。

void mexFunction(int nlhs, mxArray \*plhs[], int nrhs, const mxArray \*prhs[])   
前面提到过，Matlab的mex函数有一定的接口规范，就是指这  
nlhs：输出参数数目   
plhs：指向输出参数的指针   
nrhs：输入参数数目

例如，使用  
[a,b] = test(c,d,e)  
调用mex函数test时，传给test的这四个参数分别是  
 2，plhs，3，prhs  
其中：   
prhs[0]=c   
prhs[1]=d   
prhs[2]=e

当函数返回时，将会把你放在plhs[0]，plhs[1]里的地址赋给a和b，达到返回数据的目的。

细心的你也许已经注意到，prhs[i]和plhs[i]都是指向类型mxArray类型数据的指针。 这个类型是在mex.h中定义的，事实上，在Matlab里大多数数据都是以这种类型存在。当然还有其他的数据类型，可以参考Apiguide.pdf里的介绍。

为了让大家能更直观地了解参数传递的过程，我们把hello.c改写一下，使它能根据输   
入参数的变化给出不同的屏幕输出：

//hello.c 2.0   
#include "mex.h"   
void mexFunction(int nlhs, mxArray \*plhs[], int nrhs, const mxArray \*prhs[])   
{  
int i;   
i=mxGetScalar(prhs[0]);   
if(i==1)   
 mexPrintf("hello,world!\n");   
else   
 mexPrintf("大家好！\n");   
}

将这个程序编译通过后，执行hello(1),屏幕上会打出：   
 hello,world!   
而hello(0)将会得到：   
 大家好！   
现在，程序hello已经可以根据输入参数来给出相应的屏幕输出。在这个程序里，除了用到了屏幕输出函数mexPrintf（用法跟c里的printf函数几乎完全一样）外，还用到了一个函数：mxGetScalar，调用方式如下：   
 i=mxGetScalar(prhs[0]);   
"Scalar"就是标量的意思。在Matlab里数据都是以数组的形式存在的，mxGetScalar的作用就是把通过prhs[0]传递进来的mxArray类型的指针指向的数据（标量）赋给C程序里的变量。这个变量本来应该是double类型的，通过强制类型转换赋给了整形变量i。既然有标量，显然还应该有矢量，否则矩阵就没法传了。看下面的程序：

//hello.c 2.1   
#include "mex.h"   
void mexFunction(int nlhs, mxArray \*plhs[],   
int nrhs, const mxArray \*prhs[])   
{   
int \*i;   
i=mxGetPr(prhs[0]);   
if(i[0]==1)   
 mexPrintf("hello,world!\n");   
else   
 mexPrintf("大家好！\n");   
}   
这样，就通过mxGetPr函数从指向mxArray类型数据的prhs[0]获得了指向double类型的指针。  
但是，还有个问题，如果输入的不是单个的数据，而是向量或矩阵，那该怎么处理呢 ？通过mxGetPr只能得到指向这个矩阵的指针，如果我们不知道这个矩阵的确切大小，就   
没法对它进行计算。   
为了解决这个问题，Matlab提供了两个函数mxGetM和mxGetN来获得传进来参数的行数 和列数。下面例程的功能很简单，就是获得输入的矩阵，把它在屏幕上显示出来：   
//show.c 1.0   
#include "mex.h"   
#include "mex.h"   
void mexFunction(int nlhs, mxArray \*plhs[], int nrhs, const mxArray \*prhs[])   
{   
double \*data;   
int M,N;   
int i,j;   
data=mxGetPr(prhs[0]); //获得指向矩阵的指针   
M=mxGetM(prhs[0]); //获得矩阵的行数   
N=mxGetN(prhs[0]); //获得矩阵的列数   
for(i=0;i<M;i++)   
{ for(j=0;j<N;j++)   
 mexPrintf("%4.3f ",data[j\*M+i]);   
 mexPrintf("\n");   
 }  
}   
编译完成后，用下面的命令测试一下：   
 a=1:10;   
 b=[a;a+1];   
 show(a)   
 show(b)   
需要注意的是，在Matlab里，矩阵第一行是从1开始的，而在C语言中，第一行的序数为零，Matlab里的矩阵元素b(i,j)在传递到C中的一维数组大data后对应于data[j\*M+i] 。   
输入数据是在函数调用之前已经在Matlab里申请了内存的，由于mex函数与Matlab共用同一个地址空间，因而在prhs[]里传递指针就可以达到参数传递的目的。但是，输出参数却需要在mex函数内申请到内存空间，才能将指针放在plhs[]中传递出去。由于返回指针类型必须是mxArray，所以Matlab专门提供了一个函数：mxCreateDoubleMatrix来实现内存的申请，函数原型如下：   
 mxArray \*mxCreateDoubleMatrix(int m, int n, mxComplexity ComplexFlag)   
 m：待申请矩阵的行数   
 n：待申请矩阵的列数   
为矩阵申请内存后，得到的是mxArray类型的指针，就可以放在plhs[]里传递回去了。但是对这个新矩阵的处理，却要在函数内完成，这时就需要用到前面介绍的mxGetPr。使用 mxGetPr获得指向这个矩阵中数据区的指针（double类型）后，就可以对这个矩阵进行各种操作和运算了。下面的程序是在上面的show.c的基础上稍作改变得到的，功能是将输   
//reverse.c 1.0   
#include "mex.h"   
void mexFunction(int nlhs, mxArray \*plhs[],   
 int nrhs, const mxArray \*prhs[])   
{   
double \*inData;   
double \*outData;   
int M,N;   
int i,j;   
inData=mxGetPr(prhs[0]);   
M=mxGetM(prhs[0]);   
N=mxGetN(prhs[0]);   
plhs[0]=mxCreateDoubleMatrix(M,N,mxREAL);   
outData=mxGetPr(plhs[0]);   
for(i=0;i<M;i++)   
 for(j=0;j<N;j++)   
 outData[j\*M+i]=inData[(N-1-j)\*M+i];   
}   
当然，Matlab里使用到的并不是只有double类型这一种矩阵，还有字符串类型、稀疏矩阵、结构类型矩阵等等，并提供了相应的处理函数。本文用到编制mex程序中最经常遇到的一些函数，其余的详细情况清参考Apiref.pdf。   
通过前面两部分的介绍，大家对参数的输入和输出方法应该有了基本的了解。具备了这些知识，就能够满足一般的编程需要了。但这些程序还有些小的缺陷，以前面介绍的re由于前面的例程中没有对输入、输出参数的数目及类型进行检查，导致程序的容错性很差，以下程序则容错性较好  
#include "mex.h"   
void mexFunction(int nlhs, mxArray \*plhs[], int nrhs, const mxArray \*prhs[])   
{   
double \*inData;   
double \*outData;   
int M,N;   
//异常处理   
//异常处理   
if(nrhs!=1)   
 mexErrMsgTxt("USAGE: b=reverse(a)\n");   
 if(!mxIsDouble(prhs[0]))   
 mexErrMsgTxt("the Input Matrix must be double!\n");   
 inData=mxGetPr(prhs[0]);   
 M=mxGetM(prhs[0]);   
 N=mxGetN(prhs[0]);   
 plhs[0]=mxCreateDoubleMatrix(M,N,mxREAL);   
 outData=mxGetPr(plhs[0]);   
 for(i=0;i<M;i++)   
 for(j=0;j<N;j++)   
 outData[j\*M+i]=inData[(N-1-j)\*M+i];   
 }   
在上面的异常处理中，使用了两个新的函数：mexErrMsgTxt和mxIsDouble。MexErrMsgTxt在给出出错提示的同时退出当前程序的运行。MxIsDouble则用于判断mxArray中的数据是否double类型。当然Matlab还提供了许多用于判断其他数据类型的函数，这里不加详述。   
需要说明的是，Matlab提供的API中，函数前缀有mex-和mx-两种。带mx-前缀的大多是对mxArray数据进行操作的函数，如mxIsDouble,mxCreateDoubleMatrix等等。而带mx前缀的则大多是与Matlab环境进行交互的函数，如mexPrintf，mxErrMsgTxt等等。了解了这一点，对在Apiref.pdf中查找所需的函数很有帮助。  
至此为止，使用C编写mex函数的基本过程已经介绍完了。