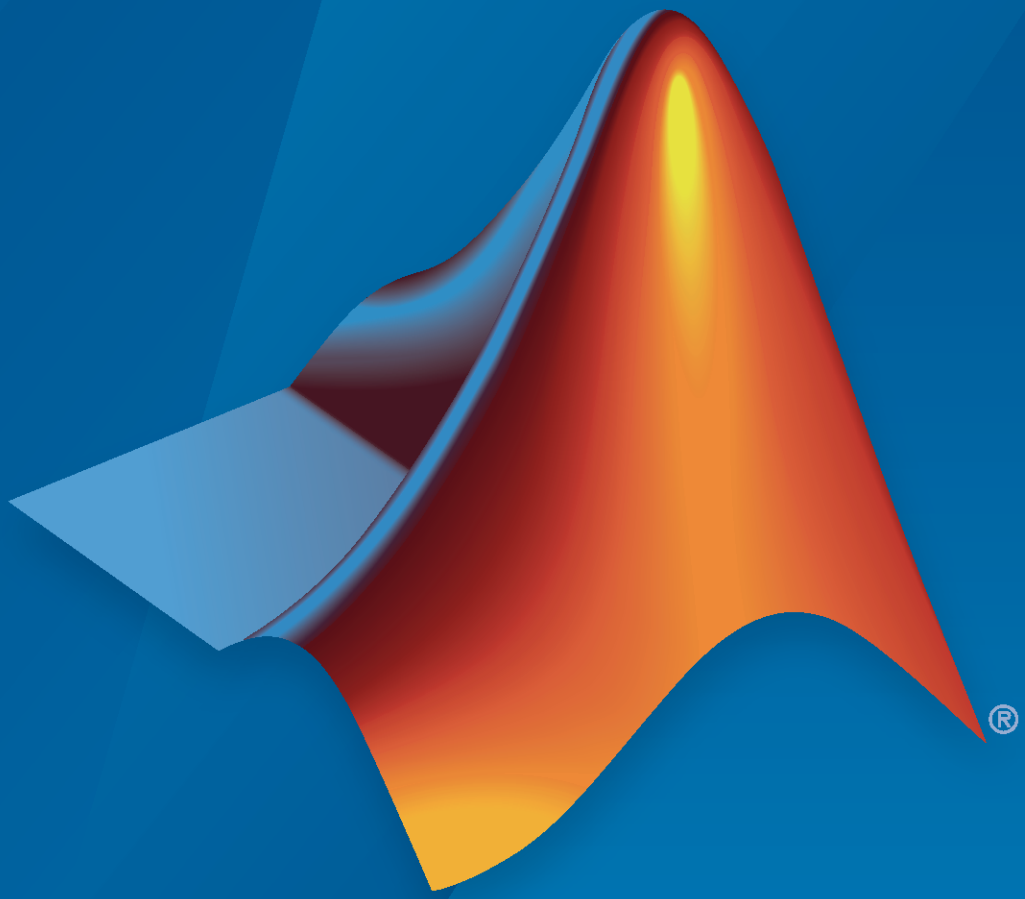
MATLAB®
外部接口



MATLAB®

R2022b

MathWorks®

如何联系 MathWorks

最新动态: **www.mathworks.com**

销售和服务: **www.mathworks.com/sales_and_services**

用户社区: **www.mathworks.com/matlabcentral**

技术支持: **www.mathworks.com/support/contact_us**

电话: 010-59827000

迈斯沃克软件 (北京) 有限公司
北京市朝阳区望京东园四区 6 号楼
北望金辉大厦 16 层 1604

MATLAB® 外部接口

© COPYRIGHT 1984–2022 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

商标

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See **www.mathworks.com/trademarks** for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

专利

MathWorks products are protected by one or more U.S. patents. Please see **www.mathworks.com/patents** for more information.

修订历史记录

1996 年 12 月	第一次印刷	MATLAB 5（版本 8）中的新增内容
1997 年 7 月	仅限在线版本	MATLAB 5.1（版本 9）中的修订内容
1998 年 1 月	第二次印刷	MATLAB 5.2（版本 10）中的修订内容
1998 年 10 月	第三次印刷	MATLAB 5.3（版本 11）中的修订内容
2000 年 11 月	第四次印刷	MATLAB 6.0（版本 12）中的修订内容和重命名内容
2001 年 6 月	仅限在线版本	MATLAB 6.1（版本 12.1）中的修订内容
2002 年 7 月	仅限在线版本	MATLAB 6.5（版本 13）中的修订内容
2003 年 1 月	仅限在线版本	MATLAB 6.5.1（版本 13 SP1）中的修订内容
2004 年 6 月	仅限在线版本	MATLAB 7.0（版本 14）中的修订内容
2004 年 10 月	仅限在线版本	MATLAB 7.0.1（版本 14SP1）中的修订内容
2005 年 9 月	仅限在线版本	MATLAB 7.1（版本 14SP3）中的修订内容
2006 年 3 月	仅限在线版本	MATLAB 7.2（版本 2006a）中的修订内容
2006 年 9 月	仅限在线版本	MATLAB 7.3（版本 2006b）中的修订内容
2007 年 3 月	仅限在线版本	MATLAB 7.4（版本 2007a）中的修订内容
2007 年 9 月	仅限在线版本	MATLAB 7.5（版本 2007b）中的修订内容
2008 年 3 月	仅限在线版本	MATLAB 7.6（版本 2008a）中的修订内容
2008 年 10 月	仅限在线版本	MATLAB 7.7（版本 2008b）中的修订内容
2009 年 3 月	仅限在线版本	MATLAB 7.8（版本 2009a）中的修订内容
2009 年 9 月	仅限在线版本	MATLAB 7.9（版本 2009b）中的修订内容
2010 年 3 月	仅限在线版本	MATLAB 7.10（版本 2010a）中的修订内容
2010 年 9 月	仅限在线版本	MATLAB 7.11（版本 2010b）中的修订内容
2011 年 4 月	仅限在线版本	MATLAB 7.12（版本 2011a）中的修订内容
2011 年 9 月	仅限在线版本	MATLAB 7.13（版本 2011b）中的修订内容
2012 年 3 月	仅限在线版本	MATLAB 7.14（版本 2012a）中的修订内容
2012 年 9 月	仅限在线版本	MATLAB 8.0（版本 2012b）中的修订内容
2013 年 3 月	仅限在线版本	MATLAB 8.1（版本 2013a）中的修订内容
2013 年 9 月	仅限在线版本	MATLAB 8.2（版本 2013b）中的修订内容
2014 年 3 月	仅限在线版本	MATLAB 8.3（版本 2014a）中的修订内容
2014 年 10 月	仅限在线版本	MATLAB 8.4（版本 2014b）中的修订内容
2015 年 3 月	仅限在线版本	MATLAB 8.5（版本 2015a）中的修订内容
2015 年 9 月	仅限在线版本	MATLAB 8.6（版本 2015b）中的修订内容
2016 年 3 月	仅限在线版本	MATLAB 9.0（版本 2016a）中的修订内容
2016 年 9 月	仅限在线版本	MATLAB 9.1（版本 2016b）中的修订内容
2017 年 3 月	仅限在线版本	MATLAB 9.2（版本 2017a）中的修订内容
2017 年 9 月	仅限在线版本	MATLAB 9.3（版本 2017b）中的修订内容
2018 年 3 月	仅限在线版本	MATLAB 9.4（版本 2018a）中的修订内容
2018 年 9 月	仅限在线版本	MATLAB 9.5（版本 2018b）中的修订内容
2019 年 3 月	仅限在线版本	MATLAB 9.6（版本 2019a）中的修订内容
2019 年 9 月	仅限在线版本	MATLAB 9.7（版本 2019b）中的修订内容
2020 年 3 月	仅限在线版本	MATLAB 9.8（版本 2020a）中的修订内容
2020 年 9 月	仅限在线版本	MATLAB 9.9（版本 2020b）中的修订内容
2021 年 3 月	仅限在线版本	MATLAB 9.10（版本 2021a）中的修订内容
2021 年 9 月	仅限在线版本	MATLAB 9.11（版本 2021b）中的修订内容
2022 年 3 月	仅限在线版本	MATLAB 9.12（版本 2022a）中的修订内容
2022 年 9 月	仅限在线版本	MATLAB 9.13（版本 2022b）中的修订内容

目录

1 外部编程语言和系统

2 外部数据接口 (EDI)

3 通过 MATLAB 使用 Java 库

4 读取和写入用 C/C++ 和 Fortran 编写的 MATLAB MAT 文件

5 从 MATLAB 调用 C++ 共享库中的函数

6 从 MATLAB 中调用 C 共享库中的函数

8 C/C++ MEX 文件

9 C++ MEX 应用程序

10 Fortran MEX 文件

11 通过 C/C++ 和 Fortran 程序调用 MATLAB Engine

12 Engine API for Java

13 MATLAB Engine for Python 主题

15 通过 MATLAB 使用 .NET 库

16 Engine API for .NET

17 通过 MATLAB 使用 COM 对象

18 MATLAB COM 客户端支持

19 MATLAB COM 自动化服务器支持

20 将 Web 服务与 MATLAB 结合使用

21 Python 接口主题

22 系统命令

23

1

外部编程语言和系统

将 MATLAB 与外部编程语言和系统集成

MATLAB 可与其他编程语言进行灵活的双向集成，从而使您能够重用原有代码。有关编程语言和支持版本的列表，请参阅 MATLAB 支持的与其他语言的接口。

从 MATLAB 中调用 C/C++ 代码

MATLAB 提供了以下功能，可帮助您将 C/C++ 算法集成到 MATLAB 应用程序中。

- C/C++ 共享库接口是应用程序在运行时动态加载的函数集合。使用共享库的优点是可以将多个库函数打包到一个接口中。此外，MATLAB 还可以管理数据类型转换。
  - 调用 C++ 库函数 - 要调用 C++ 共享库中的函数，请使用 “从 MATLAB 中调用 C++” 中所述的 **clib** 程序包。
  - 尽可能选择 C++ 接口而不是纯 C 接口。有关 C++ 支持的信息，请参阅这些限制。
  - 要调用 C 共享库中的函数，请使用 **calllib** 函数。有关信息，请参阅 “从 MATLAB 中调用 C”。此功能最适合只包含 C 代码的库，但存在限制。

  如果要更好地控制数据转换和内存管理，可以考虑编写一个 MEX 文件。
- MEX 文件是 C/C++ 算法的包装程序代码，可处理从 MATLAB 数据类型到 C 类型的转换。相对于通过 MATLAB 共享库接口来调用函数，MEX 文件性能更优。此外，MEX 文件还能让您对数据转换和内存管理实现更多的编程控制。
  - “编写可从 MATLAB（MEX 文件）调用的 C++ 函数” 使用现代 C++ 编程功能，并尽可能共享数据副本。
  - “编写可从 MATLAB（MEX 文件）调用的 C 函数”使用 “C Matrix API” 并支持现有 MEX 函数。MathWorks 建议尽可能选择 C++ MEX 文件应用程序，而不是 C MEX。但是，如果您的 MEX 函数必须在 MATLAB R2017b 或更早版本中运行，则使用 C 矩阵库编写 MEX 函数。
  - 如果一个库中有多个函数或者没有性能问题，可以考虑编写 C++ 库接口。

这些功能要求您具备一定的 C/C++ 编程技能，以创建库接口或编写 MEX 函数。但是，您可以将生成的库或 MEX 函数提供给任何 MATLAB 用户。最终用户可以像调用任何 MATLAB 函数一样调用这些功能，而无需了解 C/C++ 语言实现的基础背景知识。

要从 C/C++ 语言程序调用 MATLAB，请参阅“从 C++ 调用 MATLAB” 或“从 C 调用 MATLAB”。

在 MATLAB 中使用来自其他编程语言的对象

如果您有其他编程语言的函数和对象，可以从 MATLAB 中调用它们。要将这些对象集成到 MATLAB 应用程序中，您并不一定要具备软件开发技能。但是，您需要具有第三方库文档的访问权限。

MATLAB 支持调用以下语言的函数，也支持使用以下语言的对象。

- “从 MATLAB 中调用 C++”
- “从 MATLAB 中调用 C”
- 适用于 C/C++ 和 Fortran 的 MEX 文件函数
- “从 MATLAB 中调用 Java”
- “从 MATLAB 中调用 Python”
- “从 MATLAB 调用 .NET”

- “在 MATLAB 中使用 COM 对象”

从另一种编程语言中调用 MATLAB

您可以使用引擎应用程序从另一种语言中调用 MATLAB。使用 MATLAB 引擎 API，您可从自己的应用程序中调用 MATLAB 函数。MATLAB 具有适用于以下语言的 API。

- 用于 C++ 的引擎 API
- 适用于 Java 语言的引擎 API
- 适用于 Python 语言的引擎 API
- 适用于 C 语言的引擎 API
- 适用于 Fortran 语言的引擎 API

要创建引擎应用程序，请安装 MATLAB 支持的编译器，并使用 `mex` 命令编译应用程序。

将您的函数作为 MATLAB 函数来调用

您可以编写自己的函数，并使用 MEX API 将它们作为 MATLAB 函数进行调用。有关详细信息，请参阅“选择 MEX 应用程序” （第 7-3 页）。您可以编写以下语言的 MEX 函数。

- C++ MEX API
- C MEX API
- Fortran MEX API

要创建 MEX 文件，请安装 MATLAB 支持的编译器，并使用 `mex` 命令编译函数。

与 Web 服务通信

您可以从 MATLAB 中与 Web 服务进行通信。

- MATLAB RESTful Web 服务函数允许非编程人员使用 HTTP GET 和 POST 方法访问许多 Web 服务。
- 对于 RESTful Web 服务函数不支持的功能，可以使用 HTTP 接口类来编写自定义的 Web 访问应用程序。
- 如果您的 Web 服务基于 Web 服务描述语言 (WSDL) 文档技术，则可以使用 MATLAB WSDL 函数。

另请参阅

详细信息

- MATLAB 支持的与其他语言的接口
- 支持和兼容的编译器

外部数据接口 (EDI)

3

通过 MATLAB 使用 Java 库

Java 类路径

要在 MATLAB 中使用 Java 类，请将它们置于 Java 类路径。类路径是一系列文件和文件夹设定。加载 Java 类时，MATLAB 按照文件和文件夹在类路径中出现的顺序搜索文件和文件夹。当 MATLAB 找到包含类定义的文件时，搜索结束。

内置的 Java 类包（Java 标准库中的类）已在类路径中。您不需要修改路径即可访问这些类。

要从 MATLAB 访问 Java 类，请将它们添加到类路径中。有关信息和示例，请参阅“Static Path of Java Class Path”。

- JAR 文件类
- 包
- 单个（未打包的）类

MATLAB 将 Java 类路径分为静态路径和动态路径。MATLAB 先搜索静态路径，再搜索动态路径。

- 使用静态路径作为加载 Java 类的默认路径。
- 开发您自己的 Java 类时使用动态路径。您可在 MATLAB 会话期间随时修改和加载动态路径。

Java 类路径选项	操作
显示类路径	调用 **javaclasspath** 函数。
将文件添加到静态路径	在预设文件夹中创建名为 **javaclasspath.txt** 的 ASCII 文本文件。有关信息和示例，请参阅“Static Path of Java Class Path”。
在动态路径上添加或删除文件	调用 **javaclasspath**、**javaaddpath** 或 **javarmpath** 函数。这些函数会清除工作区中的所有现有变量和全局变量。有关详细信息，请参阅“Dynamic Path of Java Class Path”。
扩充本机方法库的搜索路径。	在预设文件夹中创建名为 **javalibrarypath.txt** 的 ASCII 文本文件。有关信息，请参阅“Locate Native Method Libraries”。

另请参阅

javaclasspath

相关示例

- “Call Java Method”
- “Call Method in Your Own Java Class”

详细信息

- “Static Path of Java Class Path”
- “Dynamic Path of Java Class Path”
- “Locate Native Method Libraries”

将数据传递给 Java 方法

MATLAB 类型到 Java 类型的映射

当您将 MATLAB 数据作为参数传递给 Java 方法时，MATLAB 会将数据转换为最适合在 Java 语言中表达该数据的类型。有关将数据传递给 **java.lang** 类型的参数时的类型映射的信息，请参阅“传递 Java 对象” （第 3-4 页）。

下表中的每行显示一种 MATLAB 类型，后跟可能的 Java 参数匹配项，其接近度从左至右递减。MATLAB 类型（元胞数组除外）可以是标量（1×1）数组或矩阵。Java 类型可以是标量值或数组。

MATLAB 参数	**Java 参数类型（标量或数组） 对象之外的类型 最接近的类型 <——————————————————————> 最不接近的类型**						
logical	**boolean**	**byte**	**short**	**int**	**long**	**float**	**double**
double	**double**	**float**	**long**	**int**	**short**	**byte**	**boolean**
single	**float**	**double**					
uint8 **int8**	**byte**	**short**	**int**	**long**	**float**	**double**	
uint16 **int16**	**short**	**int**	**long**	**float**	**double**		
uint32 **int32**	**int**	**long**	**float**	**double**			
uint64 **int64**	**long**	**float**	**double**				
string 标量、 字符向量、 **char** 标量	**String**						
string 数组、 字符向量元胞数组 请参阅“传递字符串参数” （第 3-4 页）。	**String[]**						
jClass 类型的 Java 对象	**jClass** 类型的 Java **Object**	**jClass** 的任何超类					
对象元胞数组	**Object[]**						
MATLAB 对象	不支持						

数组维度如何影响转换

维度表示数组元素寻址所需的下标数。例如，5×1 数组的维度为 1，因为您使用一个数组下标对各元素进行索引。

在将 MATLAB 转换为 Java 数组的过程中，MATLAB 以特殊方式处理维度。对于 MATLAB 数组，维度是数组中非单一维的数量。例如，10×1 数组的维度为 1，而 1×1 数组的维度为 0，因为您不能对标量值进行索引。在 Java 代码中，嵌套数组的数量决定维度。例如，**double[][]** 的维度为 2，**double** 的维度为 0。

如果 Java 数组的维数与 MATLAB 数组 **n** 中的维数匹配，则转换后的 Java 数组具有 **n** 个维度。如果 Java 数组的维数小于 **n**，则转换会从第一个单一维开始丢弃单一维。当剩余维度的数量与 Java 数组中的维度数量匹配时，转换便会停止。如果 Java 数组的维度大于 n，则 MATLAB 会添加尾部单一维。

将数字转换为整数参数

将整数类型传递给采用 Java 整数参数的 Java 方法时，MATLAB 转换与 Java 转换在整数类型之间是相同的。尤其当整数超出范围时，它不会适应参数类型的位数。对于超出范围的整数，MATLAB 会丢弃所有最低的 **n** 位。值 **n** 是参数类型中的位数。此转换与 MATLAB 整数类型之间的转换不同，后者会将超出范围的整数转换为由目标类型表示的最大值或最小值。

如果参数是浮点数，MATLAB 不会像 Java 那样将其转换为整数。MATLAB 会首先将浮点数转换为 64 位有符号整数，即截断小数部分。然后将该数字视为 **int64** 参数进行处理。

浮点数太大（在 -2^{63}–2^{63} 范围之外）时，无法用 64 位整数表示。在这种情况下，MATLAB 使用以下转换：

- 将 **int**、**short** 和 **byte** 参数值转换为 **0**。
- 将 **long** 参数值转换为 **java.lang.Long.MIN_VALUE**。
- 将 **Inf** 和 **-Inf** 值转换为 **-1**。
- 将 **NaN** 值转换为 **0**。

传递字符串参数

要使用定义为 **java.lang.String** 的参数调用 Java 方法，请传递一个 MATLAB 字符串或字符向量。MATLAB 将该参数转换为 Java **String** 对象。您还可以传递由 Java 方法返回的 **String** 对象。

如果方法参数是 **String** 类型的数组，则传递字符串数组或字符向量元胞数组。MATLAB 将输入转换为 **String** 对象的 Java 数组，维度按“数组维度如何影响转换” （第 3-4 页）中所述进行调整。

传递 Java 对象

要调用其中某个参数属于 Java 类（而不是 **java.lang.Object**）的方法，您必须传递一个 Java 对象，此对象是该类的一个实例。MATLAB 不支持 Java 自动装箱，即 MATLAB 类型不会自动转换为 Java **Object** 类型。例如，对于 **Double** 类型的参数，MATLAB 不会将 **double** 转换为 **java.lang.Double**。

传递 java.lang.Object

当方法接受 **java.lang.Object** 类的参数时，存在特例。由于此类是 Java 类层次结构的根，您可以在参数中传递任何类的对象。MATLAB 会自动将参数转换为最接近的 Java **Object** 类型，其中可能包含 Java 样式的自动装箱。下表显示了对应的转换。

MATLAB 参数	java.lang 包中的 Java Object
logical	Boolean
double	Double
single	Float
char 标量	Character
string 标量 非空 char 向量	String
uint8 int8	Byte
uint16 int16	Short
uint32 int32	Integer
uint64 int64	Long
string 数组（非标量） 字符向量元胞数组	String[]
Java 对象	参数不变
元胞数组	Object[]
MATLAB 对象	不支持

传递对象数组

要调用其中某个参数定义为 java.lang.Object 或 java.lang.Object 数组的方法，请传递一个 Java 数组或 MATLAB 元胞数组。MATLAB 会自动将元胞数组元素转换为其 Java 类型，如“传递 java.lang.Object” （第 3-4 页）表中所述。Java 数组是从 Java 构造函数返回的数组。您还可以使用 javaArray 函数在 MATLAB 中构造 Java 数组。

传递 Java 对象的元胞数组

要创建 Java 对象的元胞数组，请使用 MATLAB 语法 {a1,a2,...}。您可以通过常规方式使用语法 a{m,n,...} 对 Java 对象的元胞数组进行索引。例如，创建元胞数组 A：

```
a1 = java.lang.Double(100);
a2 = java.lang.Float(200);
A = {a1,a2}
```

```
A =

  1×2 cell array

    [1×1 java.lang.Double]    [1×1 java.lang.Float]
```

传递空矩阵、空值和缺失值

MATLAB 按如下方式转换空矩阵。

- 如果参数为空字符向量且参数声明为 String，则 MATLAB 传入空（不是 null）Java String 对象。

- 对于所有其他情况，MATLAB 会将空数组转换为 Java **null**。

空（长度为 **0**）Java 数组保持不变。

MATLAB 将字符串中的 **<missing>** 值转换为 **null**。

重载的方法

对 Java 对象调用重载的方法时，MATLAB 会将您传递的参数与针对方法定义的参数进行比较。在此类情形中，术语方法包含构造函数。MATLAB 会根据 Java 转换规则确定调用方法并将参数转换为 Java 类型。有关详细信息，请参阅“传递对象数组” （第 3-5 页）。

当您调用 Java 方法时，MATLAB 确保：

1 对象或类（对于静态方法）具有按该名称命名的方法。

2 该调用传递至少一个具有该名称的方法的相同数量的参数。

3 每个传递的参数都转换为针对该方法定义的 Java 类型。

如果所有这些条件均满足，则 MATLAB 会调用该方法。

在对重载的方法的调用中，如果有多个候选方法，MATLAB 将选择其参数最适合调用参数的方法。首先，MATLAB 会拒绝其参数类型与所传递参数不兼容的方法。例如，如果该方法具有 **double** 参数，则 **char** 参数不兼容。

然后，MATLAB 选择具有最高契合值（该方法的所有参数的契合值之和）的方法。每个参数的契合值等于基类型的契合度减去 MATLAB 数组维度与 Java 数组维度的差值。有关数组维度的信息，请参阅“数组维度如何影响转换” （第 3-4 页）。如果两个方法具有相同的契合度，则会选择在 Java 类中定义的第一个方法。

另请参阅

详细信息

- “处理从 Java 方法返回的数据” （第 3-7 页）

处理从 Java 方法返回的数据

如果 Java 方法返回原始数据类型，则 MATLAB 将转换数据，如“原始返回类型” （第 3-7 页）中的表所示。

如果 Java 方法签名指定 **java.lang.Object** 类型的返回数据，则 MATLAB 会转换返回的实际类型，如“java.lang.Object 返回类型” （第 3-7 页）中的表所示。

MATLAB 不会将其他 Java 对象转换为 MATLAB 类型。有关处理此数据的信息，请参阅“将 Java 对象转换为 MATLAB 类型的函数” （第 3-8 页）。

原始返回类型

MATLAB 将从 Java 方法返回的原始数据转换为最适合在 MATLAB 语言中表达该数据的类型。下表说明 MATLAB 如何转换数据。对于某些 Java 类型，MATLAB 以不同的方式处理标量和数组返回值。

Java 返回类型	生成的 MATLAB 类型 - 标量	转换后的 MATLAB 类型 - 数组
boolean	**logical**	**logical**
byte	**double**	**int8**
short	**double**	**int16**
int	**double**	**int32**
long	**double**	**int64**
float	**double**	**single**
double	**double**	**double**
char	**char**	**char**

示例

java.lang.String 方法 **toCharArray** 的签名为：

```
public char[] toCharArray()
```

对 **String** 对象调用该方法。MATLAB 将输出转换为 **char** 数组。

```
str = java.lang.String('hello');
res = str.toCharArray'
```

```
res =

  1×5 char array

hello
```

java.lang.Object 返回类型

当声明 Java 方法返回 **java.lang.Object** 类型的数据时，MATLAB 根据返回的实际类型转换其值。下表说明 MATLAB 如何转换数据。

实际 Java 类型	生成的 MATLAB 类型 - 标量
java.lang.Boolean	**logical**
java.lang.Byte	**double**
java.lang.Short	**double**
java.lang.Integer	**double**
java.lang.Long	**double**
java.lang.Float	**double**
java.lang.Double	**double**
java.lang.Character	**char**
java.lang.String	**char**

如果返回参数是 **Object** 的子类或 **Object** 的数组，则不会进行转换。该对象保持为 Java 对象。但是，如果您对返回的 **Object** 数组进行索引，则 MATLAB 将根据上表对值进行转换。有关详细信息，请参阅“Converting Object Array Elements to MATLAB Types”。

示例

请参阅 **getData** 方法的以下签名。

```
java.lang.Object getData()
```

如果 **getData** 返回 **java.lang.Integer** 对象，则 MATLAB 将值转换为 **double**。

将 Java 对象转换为 MATLAB 类型的函数

如果方法签名指定 **java.lang.Object**，MATLAB 将仅转换对象数据返回值。如果签名指定任何其他对象类型，则 MATLAB 不会转换该值。例如，MATLAB 会转换以下方法签名的返回值：

```
java.lang.Object getData()
```

但 MATLAB 不会转换以下方法的返回值：

```
java.lang.String getData()
```

要将 Java 对象数据转换为 MATLAB 数据，请按以下主题所述使用 MATLAB 函数：

- “转换为 MATLAB 数值类型” （第 3-8 页）
- “转换为 MATLAB 字符串” （第 3-9 页）
- “转换为 MATLAB 结构体” （第 3-9 页）
- “转换为 MATLAB 元胞数组” （第 3-9 页）

转换为 MATLAB 数值类型

要将 Java 数值类型转换为 MATLAB 类型，请使用 MATLAB 数值函数，如 **double**。**double** 函数采取的操作取决于您指定的对象的类。

- 如果对象是从 **java.lang.Number** 派生的类的实例，则 MATLAB 将对象转换为 MATLAB **double**。
- 如果对象不是数值类的实例，则 MATLAB 会检查 **toDouble** 方法的类定义。MATLAB 调用此方法来执行转换。

- 如果您创建自己的类，则请编写一个 **toDouble** 方法来指定您自己的类型转换。

注意 如果对象的类不是从 **java.lang.Number** 派生的，并且它不实现 **toDouble** 方法，则 **double** 函数会显示错误消息。

转换为 MATLAB 字符串

要将 **java.lang.String** 对象和数组转换为 MATLAB 字符串或字符向量，请使用 MATLAB **string** 或 **char** 函数。

如果 MATLAB 函数中指定的对象不是 **java.lang.String** 类的实例，则 MATLAB 会检查它对 **toString** 或 **toChar** 方法的类定义。如果您创建自己的类，则请编写一个 **toString** 或 **toChar** 方法来指定字符串转换。

注意 如果对象的类不是 **java.lang.String** 并且它不实现 **toChar** 方法，则 **char** 函数会显示错误消息。

转换为 MATLAB 结构体

如果 Java 类定义了字段名称，则请使用 **struct** 函数将对象数据转换为 MATLAB 结构体。

假设您调用一个返回 **java.awt.Polygon** 对象的 Java 方法。该类定义字段 **xpoints** 和 **ypoints**。为了运行此示例，请创建一个 **polygon** 变量。

```
polygon = java.awt.Polygon([14 42 98 124],[55 12 -2 62],4);
```

将对象转换为结构体并显示第三个点的 x,y 坐标。

```
pstruct = struct(polygon)
```

```
pstruct =

  struct with fields:

    npoints: 4
    xpoints: [4×1 int32]
    ypoints: [4×1 int32]
```

转换为 MATLAB 元胞数组

如果 Java 方法返回不同类型的数据，请使用 **cell** 函数将数据转换为 MATLAB 类型。将根据 “原始返回类型” （第 3-7 页）和 “java.lang.Object 返回类型” （第 3-7 页）表对生成的元胞数组的元素进行转换。

假设您调用返回 **java.lang.Double**、**java.awt.Point** 和 **java.lang.String** 类型参数的 Java 方法。为了运行此示例，请创建这些类型的变量。

```
import java.lang.* java.awt.*

% Create a Java array of double
dblArray = javaArray('java.lang.Double',1,10);
for m = 1:10
  dblArray(1,m) = Double(m * 7);
end
```

```
% Create a Java array of points
ptArray = javaArray('java.awt.Point',3);
ptArray(1) = Point(7.1,22);
ptArray(2) = Point(5.2,35);
ptArray(3) = Point(3.1,49);

% Create a Java array of strings
strArray = javaArray('java.lang.String',2,2);
strArray(1,1) = String('one');
strArray(1,2) = String('two');
strArray(2,1) = String('three');
strArray(2,2) = String('four');
```

将每个数组转换为一个元胞数组。您可以使用 MATLAB 函数中的 **cellArray**。

```
cellArray = {cell(dblArray),cell(ptArray),cell(strArray)}

cellArray =

  1×3 cell array

    {1×10 cell}    {3×1 cell}    {2×2 cell}
```

每个元胞都包含一种不同类型的数组。显示内容。

```
cellArray{1,1}      % Array of type double

ans =

  1×10 cell array

    [7]    [14]    [21]    [28]    [35]    [42]    [49]    [56]    [63]    [70]

cellArray{1,2}      % Array of type Java.awt.Point

ans =

  3×1 cell array

    [1×1 java.awt.Point]
    [1×1 java.awt.Point]
    [1×1 java.awt.Point]

cellArray{1,3}      % Array of type char array

ans =

  2×2 cell array

    'one'      'two'
    'three'    'four'
```

另请参阅

详细信息

- “将数据传递给 Java 方法” (第 3-3 页)

- “How MATLAB Represents Java Arrays”
- “Converting Object Array Elements to MATLAB Types”

Java 堆内存预设

您可以调整 MATLAB 分配给 Java 对象的内存量。

注意 大多数情况下，默认堆大小是足够的。

要调整堆大小，请执行下列操作：

1. 在**主页**选项卡上的**环境**部分中，点击 **预设**。选择 **MATLAB** > **常规** > **Java 堆内存**。
2. 使用滑块或微调框选择一个值。

   注意 增加堆大小会减少可用于在数组中存储数据的内存量。
3. 点击**确定**。
4. 重新启动 MATLAB。

如果在重新启动时您指定的内存量不可用，则 MATLAB 会将该值重置为默认值，并显示错误对话框。要重新调整该值，请重复上述步骤。

如果增加堆大小不能消除内存错误，请检查您的 Java 代码是否会导致内存泄漏。消除对不再有用的对象的引用。有关详细信息，请参阅 Troubleshooting Java SE。

读取和写入用 C/C++ 和 Fortran 编写的 MATLAB MAT 文件

在 C 或 C++ 中创建 MAT 文件

本节内容

在 C 中创建 MAT 文件

matcreat.c 示例说明如何使用库例程来创建可加载到 MATLAB 工作区中的 MAT 文件。该程序还演示如何通过检查 MAT 文件函数调用的返回值判断是否存在读取或写入失败。要查看代码，请在 MATLAB 编辑器中打开文件。

编译程序后，运行应用程序。此程序将创建一个可加载到 MATLAB 中的 MAT 文件 **mattest.mat**。要运行应用程序，请双击其图标或在系统提示符下输入 **matcreat**，具体取决于您的平台。

```
matcreat

Creating file mattest.mat...
```

要验证 MAT 文件，请在 MATLAB 命令提示符下键入：

```
whos -file mattest.mat
```

```
  Name           Size       Bytes  Class

  GlobalDouble    3x3          72  double array (global)
  LocalDouble     3x3          72  double array
  LocalString     1x43         86  char array

Grand total is 61 elements using 230 bytes
```

在 C++ 中创建 MAT 文件

matcreat.c 的 C++ 版本是 **matcreat.cpp**。在 MATLAB 编辑器中打开该文件。

另请参阅

相关示例

- “MAT 文件源代码文件表” (第 4-6 页)

读取用 C/C++ 编写的 MAT 文件

matdgns.c 示例说明如何使用库例程来读取和诊断 MAT 文件。要查看代码，请在 MATLAB 编辑器中打开文件。

编译程序后，运行应用程序。此程序将读取“在 C 或 C++ 中创建 MAT 文件” （第 4-2 页）示例所创建的 **mattest.mat** MAT 文件。要运行应用程序，请双击其图标或在系统提示符下输入 **matdgns**，具体取决于您的平台。

```
matdgns mattest.mat
Reading file mattest.mat...

Directory of mattest.mat:
GlobalDouble
LocalString
LocalDouble

Examining the header for each variable:
According to its header, array GlobalDouble has 2 dimensions
  and was a global variable when saved
According to its header, array LocalString has 2 dimensions
  and was a local variable when saved
According to its header, array LocalDouble has 2 dimensions
  and was a local variable when saved

Reading in the actual array contents:
According to its contents, array GlobalDouble has 2 dimensions
  and was a global variable when saved
According to its contents, array LocalString has 2 dimensions
  and was a local variable when saved
According to its contents, array LocalDouble has 2 dimensions
  and was a local variable when saved
Done
```

另请参阅

相关示例

- “在 Fortran 中创建 MAT 文件” （第 4-4 页）
- “MAT 文件源代码文件表” （第 4-6 页）

在 Fortran 中创建 MAT 文件

matdemo1.F 示例创建 MAT 文件 **matdemo.mat**。要查看代码，请在 MATLAB 编辑器中打开文件。

编译程序后，运行应用程序。此程序创建一个可加载到 MATLAB 中的 MAT 文件 **matdemo.mat**。要运行应用程序，请双击其图标或在系统提示符下键入 **matdemo1**，具体取决于您的平台：

```
matdemo1

Creating MAT-file matdemo.mat ...
Done creating MAT-file
```

要验证 MAT 文件，请在 MATLAB 命令提示符下键入：

```
whos -file matdemo.mat

  Name          Size         Bytes  Class    Attributes

  Numeric       3x3             72  double
  NumericGlobal 3x3             72  double   global
  String        1x33            66  char
```

注意 有关 Microsoft® Windows® 独立程序（非特定于 MAT 文件）的示例，请参阅 **matlabroot\extern\examples\eng_mat** 文件夹中的 **engwindemo.c**。

另请参阅

相关示例

- “读取用 C/C++ 编写的 MAT 文件” （第 4-3 页）
- “MAT 文件源代码文件表” （第 4-6 页）

在 Fortran 中读取 MAT 文件

matdemo2.F 示例说明如何使用库例程来读取由 **matdemo1.F** 创建的 MAT 文件并描述其内容。要查看代码，请在 MATLAB 编辑器中打开文件。

在编译程序后，查看结果。

```
matdemo2


Directory of Mat-file:
String
Numeric
Getting full array contents:
  1
Retrieved String
  With size   1-by-  33
  3
Retrieved Numeric
  With size   3-by-  3
```

另请参阅

相关示例

- “MAT 文件源代码文件表” (第 4-6 页)

MAT 文件源代码文件表

matlabroot/extern/examples/eng_mat 文件夹包含示例的 C/C++ 和 Fortran 源代码，用于说明如何使用 MAT 文件例程。这些示例创建独立程序。源代码对于 Windows、macOS 和 Linux® 系统都是相同的。

要编译代码示例，请首先将文件复制到一个可写文件夹中，例如 Windows 路径上的 **c:\work**。

```
copyfile(fullfile(matlabroot,'extern','examples','eng_mat',...
'filename'), fullfile('c:','work'))
```

其中，**filename** 为源代码文件的名称。

有关编译信息，请参阅：

- “MAT-File API Library and Include Files”
- “Build on macOS and Linux Operating Systems”
- “Build on Windows Operating Systems”

示例	说明
matcreat.c	C 程序，说明如何使用库例程创建可加载到 MATLAB 中的 MAT 文件。
matcreat.cpp	**matcreat.c** 程序的 C++ 版本。
matdgns.c	C 程序，说明如何使用库例程来读取和诊断 MAT 文件。
matdemo1.F	Fortran 程序，说明如何从 Fortran 程序调用 MATLAB MAT 文件函数。
matdemo2.F	Fortran 程序，说明如何使用库例程来读取由 **matdemo1.F** 创建的 MAT 文件并描述其内容。
matimport.c	基于 **matcreat.c** 的 C 程序，在示例中用于编写独立应用程序。
matreadstructarray.c	基于 **explore.c** 的 C 程序，用于读取结构体数组的内容。
matreadcellarray.c	基于 **explore.c** 的 C 程序，用于读取元胞数组的内容。

有关使用 Matrix Library 的示例，请参阅：

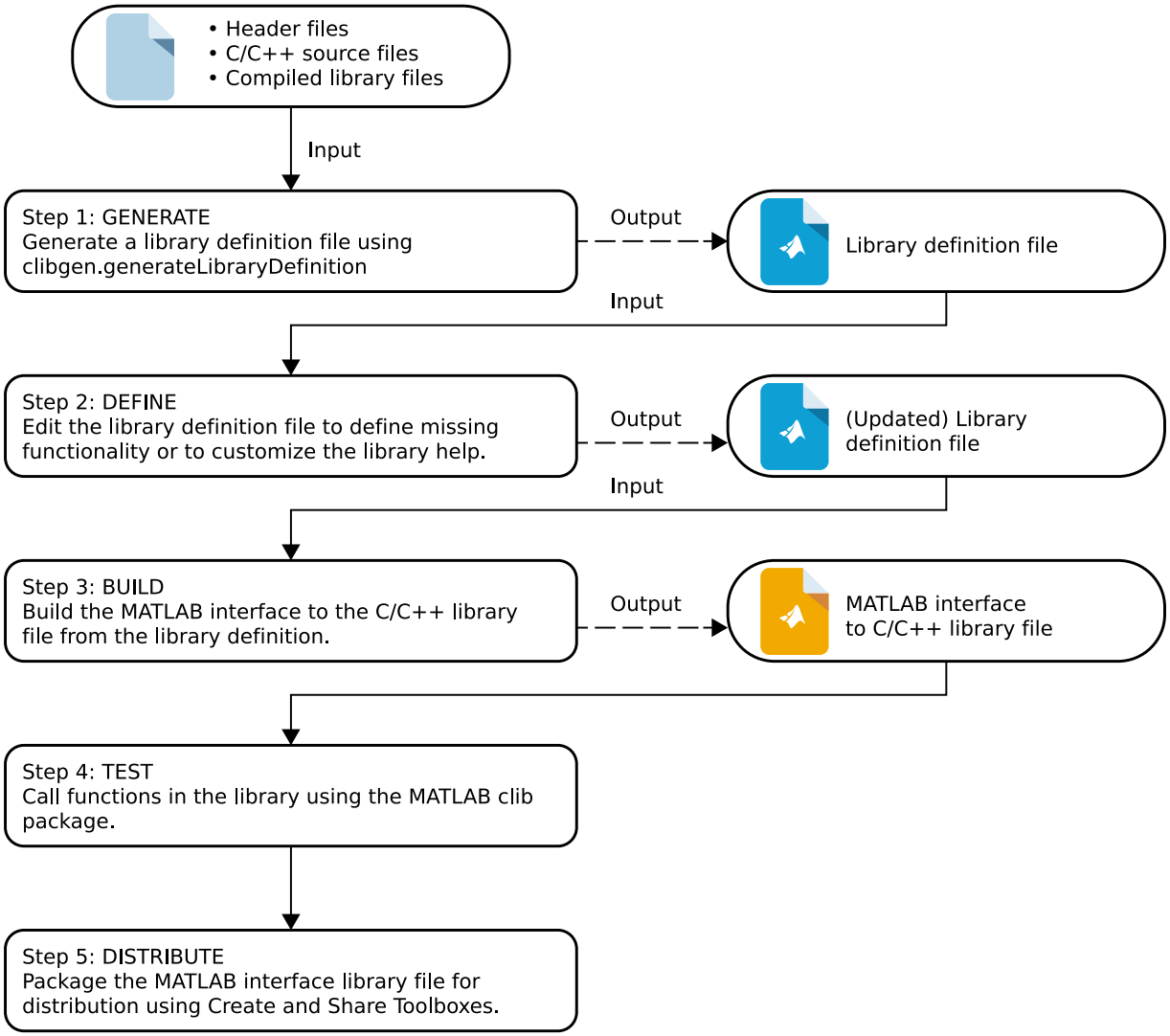
- “MEX 函数源代码示例表” （第 8-10 页）。
- “使用数据类型” （第 7-8 页）中描述的 **explore.c** 示例。

从 MATLAB 调用 C++ 共享库中的函数

将 MATLAB 接口发布到 C++ 库的步骤

有关发布接口的示例，请参阅：

- “头文件和 CPP 源文件” （第 5-10 页）
- “Windows 上的头文件和编译的库文件” （第 5-5 页）
- “Linux 上的头文件和编译的库文件” （第 5-7 页）
- “Header and C++ Compiled Library Files on macOS”
- “只包含头文件的 HPP 文件” （第 5-13 页）


• Header files
• C/C++ source files
• Compiled library files
Input
Step 1: GENERATE
Generate a library definition file using clibgen.generateLibraryDefinition
Output
Library definition file
Input
Step 2: DEFINE
Edit the library definition file to define missing functionality or to customize the library help.
Output
(Updated) Library definition file
Input
Step 3: BUILD
Build the MATLAB interface to the C/C++ library file from the library definition.
Output
MATLAB interface to C/C++ library file
Step 4: TEST
Call functions in the library using the MATLAB clib package.
Step 5: DISTRIBUTE
Package the MATLAB interface library file for distribution using Create and Share Toolboxes.

- 有关要求，请参阅“Requirements for Building Interface to C++ Libraries”。
- 对于步骤 1：生成，请参阅 **clibgen.generateLibraryDefinition**。
- 对于步骤 2：定义，请参阅“Define MATLAB Interface for C++ Library”。
- 对于步骤 3：编译，请参阅“Build C++ Library Interface and Review Contents”。
- 对于步骤 4：测试，请参阅“Call Functions in Windows Interface to C++ Compiled Library” 或“Call Functions in Linux Interface to C++ Compiled Library”。
- 对于步骤 5：分发，请参阅“Distribute MATLAB Interface to C++ Library”。

另请参阅

clibgen.generateLibraryDefinition | **build**

详细信息

- “Requirements for Building Interface to C++ Libraries”
- “Define MATLAB Interface for C++ Library”
- “Publish Help Text for MATLAB Interface to C++ Library”
- “Build C++ Library Interface and Review Contents”
- “Call Functions in C++ Compiled Library”
- “Distribute MATLAB Interface to C++ Library”
- “创建和共享工具箱”
- “Limitations to C/C++ Support”

Windows 上的头文件和编译的库文件

此示例针对 Windows 创建一个由 **matrixOperations.hpp** 定义的 C++ 库的 MATLAB 接口。MATLAB 在此文件夹中提供库文件：

```
fullfile(matlabroot, "extern", "examples", "cpp_interface")
```

使用以下步骤创建 **matrixOperations** 库的接口。

1

2 “定义缺失构造” （第 5-6 页）

3

4

另请参阅

相关示例

- “Linux 上的头文件和编译的库文件” （第 5-7 页）
- “Header and C++ Compiled Library Files on macOS”

定义缺失构造

当您在上一步中为 **matrixOperations** 库创建库定义文件时，MATLAB 指出有五个构造未完全定义。要完全定义所需功能，请编辑 **definematrixlib.m** 文件。如果您尚未打开该文件，可以点击输出消息中的链接以在编辑器中打开它。

```
C++ compiler set to 'MinGW64 Compiler (C++)'.
Definition file definematrixlib.m contains definitions for 10 constructs supported by MATLAB.
- 5 constructs are fully defined.
- 5 constructs partially defined and commented out.

To include the 5 undefined constructs in the interface, uncomment and complete the definitions in definematrixlib.m.
To build the interface, call build(definematrixlib).
```

在编辑器中打开的库定义文件中滚动，查找这些构造的注释代码块。

MATLAB 无法自动确定这些函数使用的参数大小。

- **setMat** - 类 **Mat** 的 C++ 方法
- **getMat** - 类 **Mat** 的 C++ 方法
- **copyMat** - 类 **Mat** 的 C++ 方法
- **addMat** - C++ 包函数
- **updateMatBySize** - C++ 包函数

根据 **matrixOperations** 库的文档，您可以在参数定义语句中为 **<SHAPE>** 提供值。有关详细信息，请参阅 “Define Missing SHAPE Parameter”。

1 对于每个构造，取消注释定义该构造的语句。

2 用下列值替换 **<SHAPE>** 参数。

构造	参数名称	参数 C++ 定义	描述	用下列值替换 <SHAPE>
setMat	**src**	**int [] src**	矩阵的长度由输入参数 **len** 定义。	**"len"**
getMat	**RetVal**	**int const ***	输出参数的长度由输入参数 **len** 定义。	**"len"**
copyMat	**dest**	**int * dest**	长度 **dest** 由输入参数 **len** 定义。	**"len"**
addMat	**mat**	**Mat const * mat**	该函数接受单个 **mat** 参数。	**1**
updateMatBySize	**arr**	**int * arr**	长度 **arr** 由输入参数 **len** 定义。	**"len"**

3 保存并关闭定义文件。

4 继续下一步。

Linux 上的头文件和编译的库文件

此示例为 Linux 上的 C++ 库 **matrixOperations** 创建 MATLAB 接口。该库由头文件 **matrixOperations.hpp** 和共享目标文件 **libmwmatrixOperations.so** 定义。

MATLAB 在此文件夹中提供这些文件：

```
fullfile(matlabroot,"extern","examples","cpp_interface");
```

使用下列步骤为 Linux 创建 **matrixOperations** 接口。

1

2 “定义缺失构造” （第 5-8 页）

3

4

另请参阅

相关示例

- “Windows 上的头文件和编译的库文件” （第 5-5 页）
- “Header and C++ Compiled Library Files on macOS”

定义缺失构造

当您在上一步中为 **matrixOperations** 库创建库定义文件时，MATLAB 指出有五个构造未完全定义。要完全定义所需功能，请编辑 **definematrixlib.m** 文件。如果您尚未打开该文件，可以点击输出消息中的链接以在编辑器中打开它。

```
Definition file definematrixlib.m contains definitions for 10 constructs supported by MATLAB.
- 5 constructs are fully defined.
- 5 constructs partially defined and commented out.

To include the 5 undefined constructs in the interface, uncomment and complete the definitions in definematrixlib.m.
To build the interface, call build(definematrixlib).
```

在编辑器中打开的库定义文件中滚动，查找这些构造的注释代码块。

MATLAB 无法自动确定这些函数使用的参数大小。

- **setMat** - 类 **Mat** 的 C++ 方法
- **getMat** - 类 **Mat** 的 C++ 方法
- **copyMat** - 类 **Mat** 的 C++ 方法
- **addMat** - C++ 包函数
- **updateMatBySize** - C++ 包函数

根据 **matrixOperations** 库的文档，您可以在参数定义语句中为 **<SHAPE>** 提供值。有关详细信息，请参阅 “Define Missing SHAPE Parameter” 。

1 对于每个构造，取消注释定义该构造的语句。

2 用下列值替换 **<SHAPE>** 参数。

构造	参数名称	参数 C++ 定义	描述	用下列值替换 <SHAPE>
setMat	**src**	**int [] src**	矩阵的长度由输入参数 **len** 定义。	**"len"**
getMat	**RetVal**	**int const ***	输出参数的长度由输入参数 **len** 定义。	**"len"**
copyMat	**dest**	**int * dest**	长度 **dest** 由输入参数 **len** 定义。	**"len"**
addMat	**mat**	**Mat const * mat**	该函数接受单个 **mat** 参数。	**1**
updateMatBySize	**arr**	**int * arr**	长度 **arr** 由输入参数 **len** 定义。	**"len"**

3 保存并关闭定义文件。

4 继续下一步。

定义缺失构造

当您在上一步中为 **matrixOperations** 库创建库定义文件时，MATLAB 指出有五个构造未完全定义。要完全定义所需功能，请编辑 **definematrixlib.m** 文件。如果您尚未打开该文件，可以点击输出消息中的链接以在编辑器中打开它。

```
Definition file definematrixlib.m contains definitions for 10 constructs supported by MATLAB.
- 5 constructs are fully defined.
- 5 constructs partially defined and commented out.

To include the 5 undefined constructs in the interface, uncomment and complete the definitions in definematrix
To build the interface, call build(definematrixlib).
```

在编辑器中打开的库定义文件中滚动，查找这些构造的注释代码块。

MATLAB 无法自动确定这些函数使用的参数大小。

- **setMat** - 类 **Mat** 的 C++ 方法
- **getMat** - 类 **Mat** 的 C++ 方法
- **copyMat** - 类 **Mat** 的 C++ 方法
- **addMat** - C++ 包函数
- **updateMatBySize** - C++ 包函数

根据 **matrixOperations** 库的文档，您可以在参数定义语句中为 **<SHAPE>** 提供值。有关详细信息，请参阅 “Define Missing SHAPE Parameter” 。

1 对于每个构造，取消注释定义该构造的语句。

2 用下列值替换 **<SHAPE>** 参数。

构造	参数名称	参数 C++ 定义	描述	用下列值替换 <SHAPE>
setMat	**src**	**int [] src**	矩阵的长度由输入参数 **len** 定义。	**"len"**
getMat	**RetVal**	**int const ***	输出参数的长度由输入参数 **len** 定义。	**"len"**
copyMat	**dest**	**int * dest**	长度 **dest** 由输入参数 **len** 定义。	**"len"**
addMat	**mat**	**Mat const * mat**	该函数接受单个 **mat** 参数。	**1**
updateMatBySize	**arr**	**int * arr**	长度 **arr** 由输入参数 **len** 定义。	**"len"**

3 保存并关闭定义文件。

4 继续下一步。

头文件和 CPP 源文件

此示例为在头文件 **matrixOperations.hpp** 中声明并在 C++ 源文件 **matrixOperations.cpp** 中定义的 C++ 库创建一个 MATLAB 接口。

MATLAB 在此文件夹中提供源文件：

```
fullfile(matlabroot,"extern","examples","cpp_interface");
```

使用下列步骤为 Windows 创建 **matrixOperations** 接口。

1

2 “定义缺失构造” (第 5-11 页)

3

4

另请参阅

相关示例

- “只包含头文件的 HPP 文件” (第 5-13 页)

定义缺失构造

当您在上一步中为 **matrixOperations** 库创建库定义文件时，MATLAB 指出有五个构造未完全定义。要完全定义所需功能，请编辑 **definematrixlib.m** 文件。如果您尚未打开该文件，可以点击输出消息中的链接以在编辑器中打开它。

```
C++ compiler set to 'MinGW64 Compiler (C++)'.
Definition file definematrixOperations.m contains definitions for 10 constructs supported by MATLAB.
- 5 constructs are fully defined.
- 5 constructs partially defined and commented out.

To include the 5 undefined constructs in the interface, uncomment and complete the definitions in definematrixOperations.m.
To build the interface, call build(definematrixOperations).
```

在编辑器中打开的库定义文件中滚动，查找这些构造的注释代码块。

MATLAB 无法自动确定这些函数使用的参数大小。

- **setMat** - 类 **Mat** 的 C++ 方法
- **getMat** - 类 **Mat** 的 C++ 方法
- **copyMat** - 类 **Mat** 的 C++ 方法
- **addMat** - C++ 包函数
- **updateMatBySize** - C++ 包函数

根据 **matrixOperations** 库的文档，您可以在参数定义语句中为 **<SHAPE>** 提供值。有关详细信息，请参阅 “Define Missing SHAPE Parameter”。

1 对于每个构造，取消注释定义该构造的语句。

2 用下列值替换 **<SHAPE>** 参数。

构造	参数名称	参数 C++ 定义	描述	用下列值替换 <SHAPE>
setMat	**src**	**int [] src**	矩阵的长度由输入参数 **len** 定义。	**"len"**
getMat	**RetVal**	**int const ***	输出参数的长度由输入参数 **len** 定义。	**"len"**
copyMat	**dest**	**int * dest**	长度 **dest** 由输入参数 **len** 定义。	**"len"**
addMat	**mat**	**Mat const * mat**	该函数接受单个 **mat** 参数。	**1**
updateMatBySize	**arr**	**int * arr**	长度 **arr** 由输入参数 **len** 定义。	**"len"**

3 保存并关闭定义文件。

4 继续下一步。

另请参阅

相关示例

- “Define Missing SHAPE Parameter”

只包含头文件的 HPP 文件

此示例创建名为 **school** 的 C++ 库的 MATLAB 接口。该库是在头文件 **school.hpp** 中定义的，没有编译库文件。完全由其头文件定义的库称为只包含头文件的库。

库工件	MATLAB 更新包 libname	MATLAB 帮助
头文件 **school.hpp**	**clib.school**（默认名称）	**>> doc clib.school**

此库定义表示学生和教师的类。发布此库后，MATLAB 用户可以调用 **clib.school** 包中的函数来创建 **Student** 和 **Teacher** 对象，并指定姓名和年龄。

MATLAB 在以下文件夹提供了此示例中使用的头文件：

```
fullfile(matlabroot,"extern","examples","cpp_interface");
```

使用下列步骤为 Windows 创建 **school** 接口。

1

2

3

4

另请参阅

相关示例

- “头文件和 CPP 源文件” (第 5-10 页)

6

从 MATLAB 中调用 C 共享库中的函数

调用使用 loadlibrary 加载的 C 库中的函数

共享库是应用程序在运行时动态加载的函数集合。MATLAB R2021b 及更早版本的此接口支持包含 C 头文件中定义的函数的库。要调用 C 或 C++ 库中的函数，请参阅“从 MATLAB 中调用 C++”中所述的接口。

MATLAB 在所有支持的平台上均支持动态链接。

平台	共享库	文件扩展名
Microsoft Windows	动态链接库文件	**.dll**
Linux	共享对象文件	**.so**
Apple macOS	动态共享库	**.dylib**

共享库需要头文件，后者为库中的函数提供签名。函数签名（也称为原型）用于建立函数的名称及其参数的数量和类型。指定共享库及其头文件的完整路径。

您需要安装一个 MATLAB 支持的 C 编译器。有关支持的编译器的最新列表，请参阅支持和兼容的编译器。

MATLAB 通过命令行界面访问内置于外部共享库的 C 例程。此接口允许您将外部库加载到 MATLAB 内存并访问库中的函数。尽管这两种语言环境的类型有所不同，但通常可以将类型传递到 C 函数而无需进行转换。MATLAB 将为您执行转换。

以下主题介绍了使用共享库的详细信息。

- “加载和卸载 C 共享库” （第 6-5 页）
- “查看库函数” （第 6-4 页）
- “调用库函数” （第 6-3 页）

如果库函数传递参数，则您需要确定传递到函数和从函数传递的数据类型。有关数据的信息，请参阅以下主题。

- “将参数传递给共享的 C 库函数” （第 6-7 页）
- “手动转换传递给函数的数据” （第 6-9 页）
- “在 C 共享库函数中表示指针参数” （第 6-21 页）
- “Represent Structure Arguments in C Shared Library Functions”

在完成共享库的处理后，请务必卸载库以释放内存。

另请参阅

loadlibrary | **calllib** | **libfunctions**

详细信息

- “C 和等效的 MATLAB 类型” （第 6-7 页）
- “Limitations to Shared Library Support”

调用库函数

在将共享库加载到 MATLAB 工作区后，使用 **calllib** 函数调用库中的函数。**calllib** 的语法是：

calllib('libname','funcname',arg1,...,argN)

指定库名称、函数名称，如果需要，还可以指定传递给函数的任何参数。

另请参阅

详细信息

- “将参数传递给共享的 C 库函数” （第 6-7 页）

查看库函数

要在 MATLAB 命令行窗口中显示有关库函数的信息，请使用 **libfunctions** 命令。

要查看函数签名，请使用 **-full** 开关。此选项显示用于调用以 C 语言编写的函数的 MATLAB 语法。参数列表和返回值中使用的类型是 MATLAB 类型，而不是 C 类型。有关类型的详细信息，请参阅 “C 和等效的 MATLAB 类型” （第 6-7 页）。

要在单独窗口中显示有关库函数的信息，请使用 **libfunctionsview** 函数。MATLAB 显示以下信息：

标题	说明
返回类型	方法返回的类型
名称	函数名称
参数	输入参数的有效类型

另请参阅

详细信息

- “C 和等效的 MATLAB 类型” (第 6-7 页)

加载和卸载 C 共享库

注意 要在 MATLAB R2022a 或更高版本中使用 C 库，请参阅“从 MATLAB 中调用 C++”。

为使 MATLAB 能够访问 C 共享库中的函数，请先将该库加载到内存中。加载库后，您可以请求关于库函数的信息，并直接从 MATLAB 命令行调用它们。

要将共享库加载到 MATLAB 中，请使用 **loadlibrary** 函数。最常见的语法是：

```
loadlibrary('shrlib','hfile')
```

其中，**shrlib** 是共享库文件名，**hfile** 是包含函数原型的头文件的名称。

注意 头文件为库中的函数提供签名，并且是 **loadlibrary** 的必需参数。

在完成共享库的处理后，请务必卸载库以释放内存。

另请参阅

loadlibrary | **unloadlibrary**

（错误）没有匹配的签名

如果您调用的函数没有正确的输入或输出参数，或者头文件中的函数签名中存在错误，则会发生此错误。

例如，**shrlibsample** 中 **addStructByRef** 函数的函数签名为：

```
[double, c_structPtr] addStructByRef(c_structPtr)
```

加载库。

```
addpath(fullfile(matlabroot,'extern','examples','shrlib'))
loadlibrary('shrlibsample')
```

创建一个结构体，并调用 **addStructByRef**。

```
struct.p1 = 4;
struct.p2 = 7.3;
struct.p3 = -290;
```

如果您在不带输入参数的情况下调用该函数，MATLAB 将显示错误消息。

```
[res,st] = calllib('shrlibsample','addStructByRef')
```

```
Error using calllib
No method with matching signature.
```

正确的调用是：

```
[res,st] = calllib('shrlibsample','addStructByRef',struct)
```

另请参阅

calllib | **libfunctions**

将参数传递给共享的 C 库函数

C 和等效的 MATLAB 类型

共享库接口支持所有标准标量 C 类型。下表显示了这些 C 类型及其等效的 MATLAB 类型。对于具有左列中显示的 C 类型的参数，MATLAB 使用右列中的类型。

注意 MATLAB 返回的所有标量值都是 **double** 类型。

MATLAB 原始类型

C 类型	等效的 MATLAB 类型
char、**byte**	**int8**
unsigned char、**byte**	**uint8**
short	**int16**
unsigned short	**uint16**
int	**int32**
long (Windows)	**int32**、 **long**
long (Linux)	**int64**、 **long**
unsigned int	**uint32**
unsigned long (Windows)	**uint32**、 **long**
unsigned long (Linux)	**uint64**、 **long**
float	**single**
double	**double**
char *	**char** 数组（1×n）
***char[]**	字符向量元胞数组

下表显示 MATLAB 如何将 C 指针（第 1 列）映射到等效的 MATLAB 函数签名（第 2 列）。通常，您可以将等效的 MATLAB 类型列中的变量传递给具有相应参数数据类型的函数。有关何时改用 **lib.pointer** 对象的信息，请参阅 “C 函数中的指针参数” （第 6-21 页）。

MATLAB 扩展类型

C 指针类型	参数 数据类型	等效的 MATLAB 类型	“Shared Library shrlibsample” 中的示例函数
double *	**doublePtr**	**double**	**addDoubleRef**
float *	**singlePtr**	**single**	
intsize *（整数指针类型）	**(u)int(size)Ptr** 例如，**int64 *** 变为 **int64Ptr**。	**(u)int(size)**	**multiplyShort**
byte[]	**int8Ptr**	**int8**	
char[]（以空值结尾的传值字符串）	**cstring**	**char** 数组（1×n）	**stringToUpper**
**char **（指向字符串的指针数组）	**stringPtrPtr**	字符向量元胞数组	
enum	**enumPtr**		
type **	**typePtrPtr** 例如，**double **** 变为 **doublePtrPtr**。	**lib.pointer** 对象	**allocateStruct**
void *	**voidPtr**		**deallocateStruct**
void **	**voidPtrPtr**	**lib.pointer** 对象	
struct（C 样式的结构体）	**structure**	MATLAB **struct**	**addStructFields**
mxArray *	**MATLAB array**	MATLAB 数组	
mxArray **	**MATLAB arrayPtr**	**lib.pointer** 对象	

MATLAB 如何显示函数签名

以下是有关 MATLAB 函数签名中显示的输入和输出参数的注意事项。

- 许多参数（如 **int32** 和 **double**）都与对应的 C 参数类似。在这些情况下，请传入针对这些参数所显示的 MATLAB 类型。
- 某些 C 参数（例如 ****double** 或预定义的结构体）与标准 MATLAB 类型不同。在这些情况下，请传递标准 MATLAB 类型并让 MATLAB 来进行转换，或者您自己使用 MATLAB 函数 **libstruct** 和 **libpointer** 转换数据。有关详细信息，请参阅“手动转换传递给函数的数据” （第 6-9 页）。
- C 函数常通过传引用输入参数来返回数据。MATLAB 创建其他输出参数以返回这些值。以 **Ptr** 或 **PtrPtr** 结尾的输入参数也列为输出。

有关 MATLAB 函数签名的示例，请参阅“Shared Library shrlibsample”。

传递参数的指导原则

- 非标量参数必须在库函数中声明为传引用。
- 如果库函数使用单下标索引来引用二维矩阵，请记住，C 程序会逐行处理矩阵；MATLAB 按列处理矩阵。要从函数中获取 C 行为，请在调用函数之前转置输入矩阵，然后转置函数输出。
- 使用空数组 **[]** 将 **NULL** 参数传递给支持可选输入参数的库函数。此表示法仅当参数声明为 **Ptr** 或 **PtrPtr** 时才有效，如 **libfunctions** 或 **libfunctionsview** 所示。

空指针

您可以通过以下方式创建 **NULL** 指针以传递给库函数：

- 传递空数组 **[]** 作为参数。
- 使用 **libpointer** 函数：

  ```
  p = libpointer; % no arguments

  p = libpointer('string') % string argument

  p = libpointer('cstring') % pointer to a string argument
  ```
- 使用 **libstruct** 函数：

  ```
  p = libstruct('structtype'); % structure type
  ```

空 libstruct 对象

要创建空 **libstruct** 对象，请仅使用 **structtype** 参数调用 **libstruct**。例如：

```
sci = libstruct('c_struct')
get(sci)

    p1: 0
    p2: 0
    p3: 0
```

MATLAB 显示初始化后的值。

手动转换传递给函数的数据

在大多数情况下，MATLAB 软件会自动将传入和传出外部库函数的数据转换为外部函数期望的类型。不过，您也可以选择手动转换参数数据。例如：

- 将相同数据传递给一系列库函数时，请在调用第一个函数之前手动进行一次转换，而不是在每次调用时都让 MATLAB 自动转换它。该策略可以减少不必要的复制和转换操作次数。
- 在传递大型结构体时，通过创建与函数中使用的 C 结构体形状匹配的 MATLAB 结构体（而不是使用一般 MATLAB 结构体）来节省内存。**libstruct** 函数以从库中获取的 C 结构体为模型创建 MATLAB 结构体。
- 如果外部函数的参数使用多个引用级别（例如 **double ****），则请传递使用 **libpointer** 函数创建的指针，而不是依赖 MATLAB 来自动转换类型。

另请参阅

libstruct | **libpointer** | **libfunctions** | **libfunctionsview**

相关示例

- “Shared Library shrlibsample”

详细信息

- “Represent Structure Arguments in C Shared Library Functions”

传递字符串参数示例

stringToUpper 函数

shrlibsample 库中的 **stringToUpper** 函数将输入参数中的字符转换为大写。输入参数 **char *** 是指向字符串的 C 指针。

```
EXPORTED_FUNCTION char* stringToUpper(char *input)
{
  char *p = input;

  if (p != NULL)
    while (*p!=0)
      *p++ = toupper(*p);
  return input;
}
```

下表显示了 **stringToUpper** 的函数签名。MATLAB 将 C 指针类型 (**char ***) 映射为 **cstring**，以便您将 MATLAB 字符数组传递给该函数。

返回类型	名称	参数
[cstring, cstring]	stringToUpper	(cstring)

将 MATLAB 字符数组转换为大写

此示例说明如何将 MATLAB 字符数组 **str** 传递给 C 函数 **stringToUpper**。

```
str = 'This was a Mixed Case string';
```

加载包含 **stringToUpper** 函数的库。

```
if not(libisloaded('shrlibsample'))
   addpath(fullfile(matlabroot,'extern','examples','shrlib'))
   loadlibrary('shrlibsample')
end
```

将 **str** 传递给该函数。

```
res = calllib('shrlibsample','stringToUpper',str)
```

```
res =
'THIS WAS A MIXED CASE STRING'
```

输入参数是指向 **char** 类型的指针。但是，MATLAB 字符数组不是指针，因此 **stringToUpper** 函数不会修改输入参数 **str**。

```
str
```

```
str =
'This was a Mixed Case string'
```

另请参阅

相关示例

- “Shared Library shrlibsample”
- “迭代 lib.pointer 对象” (第 6-19 页)

传递指针示例

multDoubleRef 函数

shrlibsample 库中的 **multDoubleRef** 函数将输入乘以 **5**。

```
EXPORTED_FUNCTION double *multDoubleRef(double *x)
{
    *x *= 5;
    return x;
}
```

输入是指向 **double** 的指针，而该函数也返回指向 **double** 的指针。MATLAB 函数签名是：

返回类型	名称	参数
[lib.pointer, doublePtr]	multDoubleRef	(doublePtr)

传递双精度类型的指针

此示例说明如何构造指针并传递给 C 函数 **multDoubleRef**。

加载包含该函数的库。

```
if not(libisloaded('shrlibsample'))
   addpath(fullfile(matlabroot,'extern','examples','shrlib'))
   loadlibrary('shrlibsample')
end
```

构造指向输入参数 **X** 的指针 **Xptr**。

```
X = 13.3;
Xptr = libpointer('doublePtr',X);
```

验证 **Xptr** 的内容。

```
get(Xptr)
```

```
      Value: 13.3000
   DataType: 'doublePtr'
```

调用该函数并检查结果。

```
calllib('shrlibsample','multDoubleRef',Xptr);
Xptr.Value
```

```
ans = 66.5000
```

Xptr 是一个句柄对象。此句柄的副本引用同一个底层对象，对句柄对象执行的任何操作都影响该对象的所有副本。但是，**Xptr** 不是一个 C 语言指针。虽然它指向 **X**，但并不包含 **X** 的地址。该函数修改 **Xptr** 的 Value 属性，但不修改底层对象 **X** 中的值。**X** 的原始值保持不变。

```
X
```

```
X = 13.3000
```

从现有 lib.pointer 对象创建指针偏移

此示例说明如何创建指向 MATLAB 向量 **X** 的子集的指针。仅当原始指针存在时，新指针才有效。

创建指向向量的指针。

```
X = 1:10;
xp = libpointer('doublePtr',X);
xp.Value
```

```
ans = 1×10

   1   2   3   4   5   6   7   8   9   10
```

使用 lib.pointer 加号运算符 (+) 创建指向 **X** 的最后六个元素的指针。

```
xp2 = xp + 4;
xp2.Value
```

```
ans = 1×6

   5   6   7   8   9   10
```

多级指针

多级指针是指具有多级引用的参数。MATLAB 中的多级指针类型使用后缀 **PtrPtr**。例如，使用 **doublePtrPtr** 表示 C 参数 **double ****。

当调用接受多级指针参数的函数时，可以使用 **lib.pointer** 对象并让 MATLAB 将其转换为多级指针。

allocateStruct 和 deallocateStruct 函数

shrlibsample 库中的 **allocateStruct** 函数接受 **c_structPtrPtr** 参数。

```
EXPORTED_FUNCTION void allocateStruct(struct c_struct **val)
{
   *val=(struct c_struct*) malloc(sizeof(struct c_struct));
   (*val)->p1 = 12.4;
   (*val)->p2 = 222;
   (*val)->p3 = 333333;
}
```

MATLAB 函数签名为：

返回类型	名称	参数
c_structPtrPtr	allocateStruct	(c_structPtrPtr)
voidPtr	deallocateStruct	(voidPtr)

传递多级指针

此示例说明如何将多级指针传递给 C 函数。

加载包含 **allocateStruct** 和 **deallocateStruct** 的库。

```
if not(libisloaded('shrlibsample'))
   addpath(fullfile(matlabroot,'extern','examples','shrlib'))
   loadlibrary('shrlibsample')
end
```

创建 **c_structPtr** 指针。

```
sp = libpointer('c_structPtr');
```

调用 **allocateStruct** 以便为结构体分配内存。

```
res = calllib('shrlibsample','allocateStruct',sp)
```

```
res = struct with fields:
   p1: 12.4000
   p2: 222
   p3: 333333
```

释放 **allocateStruct** 函数产生的内存。

```
calllib('shrlibsample','deallocateStruct',sp)
```

返回字符串数组

假设您有一个库 **myLib**，其中有一个函数 **acquireString**，它读取一个字符串数组。函数签名为：

返回类型	名称	参数
char**	acquireString	(void)

```
char** acquireString(void)
```

以下**伪代码**说明如何操作返回值，即指向字符串的指针数组。

```
ptr = calllib(myLib,'acquireString')
```

MATLAB 创建 **stringPtrPtr** 类型的 **lib.pointer** 对象 **ptr**。此对象指向第一个字符串。要查看其他字符串，请递增指针。例如，要显示前三个字符串，请键入：

```
for index = 0:2
   tempPtr = ptr + index;
   tempPtr.Value
end
```

```
ans =
   'str1'
ans =
   'str2'
```

```
ans =
   'str3'
```

另请参阅

libpointer

传递数组示例

print2darray 函数

shrlibsample 库中的 **print2darray** 函数显示列数为三、行数可变的二维数组的值。参数 **my2d** 是 **double** 类型的二维数组。参数 **len** 是行数。

```
EXPORTED_FUNCTION void print2darray(double my2d[][3],int len)
{
   int indxi,indxj;
   for(indxi=0;indxi<len;++indxi)
   {
      for(indxj=0;indxj<3;++indxj)
      {
         mexPrintf("%10g",my2d[indxi][indxj]);
      }
      mexPrintf("\n");
   }
}
```

将 MATLAB 数组转换为 C 样式的维度

此示例说明如何将 MATLAB® 数组中按列存储的数据传递给采用行×列格式的 C 函数。

加载包含 **print2darray** 函数的库。

```
if not(libisloaded('shrlibsample'))
   addpath(fullfile(matlabroot,'extern','examples','shrlib'))
   loadlibrary('shrlibsample')
end
```

创建一个 4 行 3 列的 MATLAB 数组。

```
m = reshape(1:12,4,3)
```

```
m = 4×3

   1   5   9
   2   6  10
   3   7  11
   4   8  12
```

显示值。第一列是 [1 4 7 10] 而不是 [1 2 3 4]。

```
calllib('shrlibsample','print2darray',m,4)
```

```
      1       2       3
      4       5       6
      7       8       9
     10      11      12
```

转置 **m** 以获得所需的结果。

```
calllib('shrlibsample','print2darray',m',4)
```

```
     1     5     9
     2     6    10
     3     7    11
     4     8    12
```

multDoubleArray 函数

shrlibsample 库中的 **multDoubleArray** 函数将数组中的每个元素乘以三。该函数使用单一下标（线性索引）来导航输入数组。

```
EXPORTED_FUNCTION void multDoubleArray(double *x,int size)
{
   /* Multiple each element of the array by 3 */
   int i;
   for (i=0;i<size;i++)
   *x++ *= 3;
}
```

MATLAB 函数签名是：

返回类型	名称	参数
doublePtr	**multDoubleArray**	**(doublePtr, int32)**

保留三维 MATLAB 数组

此示例说明 C 函数如何更改 MATLAB® 数组的维度，以及如何还原它的形状。

加载库。

```
if not(libisloaded('shrlibsample'))
   addpath(fullfile(matlabroot,'extern','examples','shrlib'))
   loadlibrary('shrlibsample')
end
```

创建一个 2×5×2 输入数组并显示其维度。

```
vin = reshape(1:20,2,5,2);
vs = size(vin)
```

```
vs = 1×3

     2     5     2
```

调用 **multDoubleArray** 以对每个元素进行乘法运算。显示输出的维度。

```
vout = calllib('shrlibsample','multDoubleArray',vin,20);
size(vout)
```

```
ans = 1×2

     2    10
```

还原初始形状：

```
vout = reshape(vout,vs);
size(vout)
```

```
ans = 1×3

     2     5     2
```

迭代 lib.pointer 对象

从 lib.pointer 对象创建元胞数组

此示例说明如何从 **getListOfStrings** 函数的输出创建 MATLAB® 字符向量元胞数组 **mlStringArray**。

加载 **shrlibsample** 库。

```
if not(libisloaded('shrlibsample'))
   addpath(fullfile(matlabroot,'extern','examples','shrlib'))
   loadlibrary('shrlibsample')
end
```

调用 **getListOfStrings** 函数以创建一个字符向量数组。该函数返回指向该数组的指针。

```
ptr = calllib('shrlibsample','getListOfStrings');
class(ptr)
```

```
ans =
'lib.pointer'
```

创建索引变量以循环访问数组。对函数返回的数组使用 **ptrindex**，对 MATLAB 数组使用 **index**。

```
ptrindex = ptr;
index = 1;
```

创建字符向量元胞数组 **mlStringArray**。将 **getListOfStrings** 的输出复制到该元胞数组。

```
% read until end of list (NULL)
while ischar(ptrindex.value{1})
   mlStringArray{index} = ptrindex.value{1};
   % increment pointer
   ptrindex = ptrindex + 1;
   % increment array index
   index = index + 1;
end
```

查看元胞数组的内容。

```
mlStringArray
```

```
mlStringArray = 1x4 cell
   {'String 1'}   {'String Two'}   {0x0 char}   {'Last string'}
```

对结构体数组执行指针算术

此示例说明如何使用指针算术来访问结构体的元素。该示例基于 **shrlibsample.h** 头文件中的 **c_struct** 定义创建一个 MATLAB 结构体。

加载该定义。

```
if not(libisloaded('shrlibsample'))
   addpath(fullfile(matlabroot,'extern','examples','shrlib'))
```

```
    loadlibrary('shrlibsample')
end
```

创建 MATLAB 结构体。

```
s = struct('p1',{1,2,3},'p2',{1.1,2.2,3.3},'p3',{0});
```

创建指向该结构体的指针。

```
sptr = libpointer('c_struct',s);
```

读取第一个元素的值。

```
v1 = sptr.Value
```

```
v1 = struct with fields:
    p1: 1
    p2: 1
    p3: 0
```

通过递增指针来读取下一个元素的值。

```
sptr = sptr + 1;
v2 = sptr.Value
```

```
v2 = struct with fields:
    p1: 2
    p2: 2
    p3: 0
```

在 C 共享库函数中表示指针参数

C 函数中的指针参数

外部库中的许多函数通过引用传递参数。当通过引用传递参数时，传递的是值的指针。在函数签名中，指针参数的名称以 **Ptr** 和 **PtrPtr** 结尾。尽管 MATLAB 不支持通过引用传递参数，但您可以创建一个称为 lib.pointer 对象并且与 C 指针兼容的 MATLAB 参数。该对象是 MATLAB **lib.pointer** 类的实例。

通常，您只需要传递 MATLAB 变量（传值参数）即可，即使该函数的签名声明参数为指针时亦如此。但有时传递 **lib.pointer** 非常有用。

- 需要修改输入参数中的数据。
- 正在传递大量数据，并且需要控制 MATLAB 复制数据的时间。
- 库将存储和使用指针，因此，您希望 MATLAB 函数控制 **lib.pointer** 对象的生存时间。

将字符串置入 Void 指针

C 将字符表示为 8 位整数。要将 MATLAB 字符数组用作输入参数，请将字符串转换为适当的类型并创建一个 **voidPtr**。例如：

```
str = 'string variable';
vp = libpointer('voidPtr',[int8(str) 0]);
```

语法 **[int8(str) 0]** 会创建 C 函数所需的以空值结尾的字符串。要读取该字符串并验证指针类型，请输入：

```
char(vp.Value)
vp.DataType
```

```
ans =
string variable
ans =
voidPtr
```

当外部函数原型将参数定义为指针时，MATLAB 会自动将传值参数转换为传引用参数。使用以下语法可调用一个将指向字符串的 **voidPtr** 作为输入参数的函数。

```
func_name([int8(str) 0])
```

尽管 MATLAB 会将该参数从值转换为指针，但它必须是正确的类型。

外部库的内存分配

一般而言，MATLAB 会在您每次将变量传递到库函数时传递一个有效的内存地址。如果库存储指针并在一段时间内访问缓冲区，则使用 **lib.pointer** 对象。在这类情况下，请确保 MATLAB 能够控制缓冲区的生存时间，并防止创建多个数据副本。以下**伪代码**是一个异步数据采集的示例，其中显示了在这种情形下如何使用 **lib.pointer**。

假设外部库 **myLib** 具有以下函数：

```
AcquireData(int points,short *buffer)
IsAquisitionDone(void)
```

其中，**buffer** 的声明如下：

```
short buffer[99]
```

首先，创建一个 **lib.pointer**，指向包含 99 个点的数组：

```
BufferSize = 99;
pBuffer = libpointer('int16Ptr',zeros(BufferSize,1));
```

然后，开始采集数据并在循环中等待，直至完成数据采集：

```
calllib('myLib','AcquireData,BufferSize,pbuffer)
while (~calllib('myLib','IsAcquisitionDone')
  pause(0.1)
end
```

以下语句将读取缓冲区中的数据：

```
result = pBuffer.Value;
```

当库完成缓冲区操作后，清除 MATLAB 变量：

```
clear pBuffer
```

另请参阅

lib.pointer

探索 libstruct 对象

此示例说明如何显示和修改 libstruct 对象 **c_struct** 的信息。

加载包含 **c_struct** 定义的 **shrlibsample** 库。

```
if not(libisloaded('shrlibsample'))
   addpath(fullfile(matlabroot,'extern','examples','shrlib'))
   loadlibrary('shrlibsample')
end
```

创建 **libstruct** 对象。对象 **sc** 是名为 **lib.c_struct** 的 MATLAB® 类的一个实例。

```
sc = libstruct('c_struct')
```

```
sc =

   lib.c_struct
```

设置结构体字段值。

```
set(sc,'p1',100,'p2',150,'p3',200)
```

显示字段值。

```
get(sc)
```

```
   p1: 100
   p2: 150
   p3: 200
```

使用 MATLAB 字段结构体语法修改值。

```
sc.p1 = 23;
get(sc)
```

```
   p1: 23
   p2: 150
   p3: 200
```

MEX 文件简介

MEX 文件函数

MEX 文件是在 MATLAB 中创建的函数，可用来调用 C/C++ 程序或 Fortran 子例程。MEX 函数的行为类似于 MATLAB 脚本或函数。

要调用 MEX 函数，请使用 MEX 文件的名称，但不带文件扩展名。MEX 文件只包含一个函数或子例程。调用语法取决于 MEX 函数定义的输入和输出参数。MEX 文件必须位于您的 MATLAB 路径下。

虽然 MATLAB 脚本和函数的扩展名 **.m** 和 **.mlx** 独立于平台，但 MEX 函数具有如下所示的 64 位平台特定的文件扩展名：

- Linux - **.mexa64**
- Apple macOS - **.mexmaci64**
- Microsoft Windows - **.mexw64**

选择 MEX 应用程序

您可以通过 MATLAB 命令行调用您自己的 C、C++ 或 Fortran 程序（就当它们是内置函数一样）。这些程序称为 MEX 函数。MEX 函数并不适用于所有的应用程序。MATLAB 是一种高生产率环境，尤其适用于消除像 C 或 C++ 等编译语言中需要花费大量时间的低级编程。一般情况下，请在 MATLAB 中进行编程。除非您的应用程序需要，否则请不要使用 MEX 文件。

要创建 MEX 函数，请使用 MATLAB API 编写您的程序，然后使用 **mex** 命令编译它。这些 API 提供以下功能：

- 从 MEX 函数中调用 MATLAB 函数。
- 无缝集成到 MATLAB 中，从 MATLAB 获取输入并向其返回结果。
- 支持 MATLAB 数据类型。

C++ MEX 函数

从 MATLAB R2018a 开始，请使用这些支持 C++11 编程功能的 API 编写您的 C++ MEX 函数。这些 API 基于 **matlab::data::Array** 类，提供更好的类型安全性、数组边界检查并支持现代 C++ 构造以简化编码。

- “C++ MEX API” （第 9-10 页）
- “MATLAB Data API for C++”

有关详细信息，请参阅“编写可从 MATLAB（MEX 文件）调用的 C++ 函数”。

MATLAB R2017b 及更早版本的 C/C++ MEX 函数

如果您的 MEX 函数必须在 MATLAB R2017b 或更早版本中运行，或如果您更喜欢使用 C 语言，请使用这些库中基于 **mxArray** 数据结构体的函数编写源文件。

- 关于 “编写可从 MATLAB（MEX 文件）调用的 C 函数”的 C MEX API
- “C Matrix API”

有关详细信息，请参阅“编写可从 MATLAB（MEX 文件）调用的 C 函数”。

小心 不要将 C 矩阵 API 中的函数与 MATLAB Data API 中的函数混合使用。

Fortran MEX 函数

要编写 Fortran MEX 函数，请使用这些基于 **mxArray** 数据结构体的 API。

- “Fortran MEX API”
- “Fortran Matrix API”

有关详细信息，请参阅“编写可从 MATLAB（MEX 文件）调用的 Fortran 函数”。

MEX 术语

MEX 表示 MATLAB 可执行文件，它具有不同含义，如下表所示。

MEX 术语	定义
源 MEX 文件	C、C++ 或 Fortran 源代码文件。
二进制 MEX 文件或 MEX 函数	在 MATLAB 环境中执行的动态链接子例程。
MEX API	C MEX API 和 Fortran MEX API 中的函数，用于在 MATLAB 环境中执行运算。
mex 编译脚本	基于源文件创建二进制文件的 MATLAB 函数。

另请参阅

mxArray | **matlab::data::Array** | **mex**

详细信息

- “MEX 函数源代码示例表” (第 8-10 页)

MATLAB 数据

MATLAB 数组

MATLAB 语言适用于单一对象类型：MATLAB 数组。所有 MATLAB 变量（包括标量、向量、矩阵、字符数组、元胞数组、结构体和对象）都以 MATLAB 数组形式存储。在 C/C++ 中，MATLAB 数组声明为 **mxArray** 类型。**mxArray** 结构体包含有关数组的以下信息：

- 数组的类型
- 数组的维度
- 与此数组相关联的数据
- 如果是数值数组，则指出变量的复实性
- 如果是稀疏数组，则提供它的索引和非零最大元素
- 如果是结构体数组或对象数组，则提供字段的数量和字段名称

要访问 **mxArray** 结构体，请使用 C 或 Fortran 矩阵 API 中的函数。这些函数允许您创建、读取和查询有关 MEX 文件中 MATLAB 数据的信息。矩阵 API 使用 **mwSize** 和 **mwIndex** 类型来避免可移植性问题，使得 MEX 源文件能在所有系统上正确编译。

mxArray 的生命周期

像 MATLAB 函数一样，MEX 文件入口例程通过引用传递 MATLAB 变量。但是，这些参数是 C 指针。指向变量的指针是该变量的地址（在内存中的位置）。MATLAB 函数自动为您处理数据存储。在将数据传递给 MEX 文件时，您需要使用指针，这些指针遵循访问和操作变量的特定规则。有关使用指针的信息，请参阅编程参考，如 Kernighan, B. W. 和 D. M. Ritchie 编写的 The C Programming Language。

注意 由于变量使用内存，您需要了解 MEX 文件如何创建 **mxArray** 以及您释放（腾出）内存的责任。这对于防止内存泄漏非常重要。**mxArray** 的生命周期以及管理内存的规则取决于它是输入参数、输出参数还是局部变量。调用哪个函数来取消分配 **mxArray** 取决于您用来创建它的函数。有关详细信息，请在 C 矩阵 API 中查找用于创建数组的函数。

输入参数 prhs

通过 **prhs** 输入参数传递给 MEX 文件的 **mxArray** 存在于 MEX 文件的范围之外。不要为 **prhs** 参数中的任何 **mxArray** 释放内存。此外，**prhs** 变量是只读的；不要在您的 MEX 文件中修改它们。

输出参数 plhs

如果为输出参数创建 **mxArray**（分配内存并创建数据），则内存和数据会超出 MEX 文件的范围。不要为 **plhs** 输出参数中返回的 **mxArray** 释放内存。

局部变量

每次您使用 **mxCreate*** 函数创建 **mxArray** 或调用 **mxCalloc** 及关联的函数时，都会分配内存。在遵守处理输入和输出参数的规则的情况下，MEX 文件应该销毁临时数组并释放动态分配的内存。要取消分配内存，请使用 **mxDestroyArray** 或 **mxFree**。有关要使用哪个函数的信息，请参阅 MX Matrix Library。

数据存储

MATLAB 采用列优先（逐列）编号方案存储数据，这是 Fortran 存储矩阵的方式。MATLAB 使用此约定，因为它最初是用 Fortran 编写的。MATLAB 首先在内部存储来自第一列的数据元素，然后继续存储来自第二列的数据元素，以此类推，直到最后一列。

例如，给定矩阵：

```
a = ['house'; 'floor'; 'porch']
```

```
a =
  house
  floor
  porch
```

其维度是：

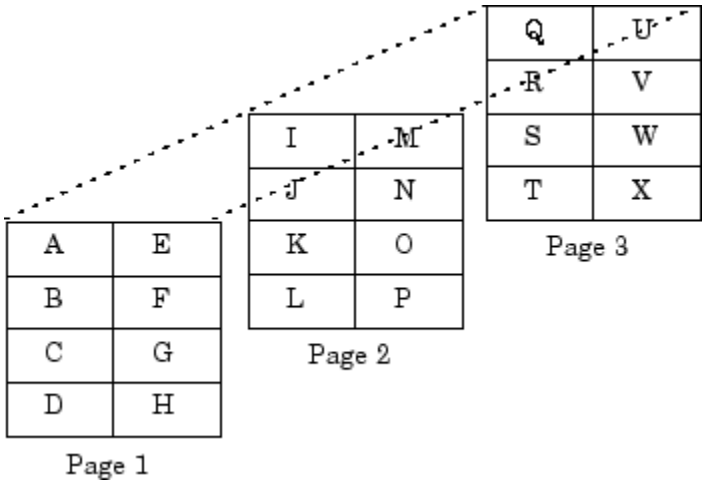
```
size(a)
```

```
ans =
   3    5
```

其数据存储为：

h	f	p	o	l	o	u	o	r	s	o	c	e	r	h

如果一个矩阵是 N 维矩阵，则 MATLAB 以 N 优先的顺序表示数据。例如，以维度为 $4\times2\times3$ 的三维数组为例。尽管您可以将数据可视化为：



MATLAB 在内部按照以下顺序表示此三维数组的数据：

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

mxCalcSingleSubscript 函数使用 N 维下标创建从数组的第一个元素到所需元素的偏移量。

MATLAB 数据类型

复数双精度矩阵

MATLAB 中最常见的数据类型是复数双精度非稀疏矩阵。这些矩阵是 **double** 类型，维度为 **m**×**n**，其中 **m** 为行数，**n** 为列数。数据存储为一个由交错式双精度数字构成的向量，其中实部和虚部彼此相邻存储。指向此数据的指针称为 **pa**（指向数组的指针）。要测试非复矩阵，请调用 **mxIsComplex**。

在 MATLAB 版本 9.4（版本 2018a）之前，MATLAB 使用分离式存储表示。数据存储为两个由双精度数值构成的向量 - 一个向量包含实数数据，另一个向量包含虚数数据。指向此数据的指针分别称为 **pr**（指向实数数据的指针）和 **pi**（指向虚数数据的指针）。非复矩阵是其 **pi** 为 **NULL** 的矩阵。但是，要测试非复矩阵，请调用 **mxIsComplex**。

其他数值矩阵

MATLAB 支持单精度浮点数以及 8、16、32 和 64 位整数，包括有符号数和无符号数。

逻辑矩阵

逻辑数据类型分别使用数字 **1** 和 **0** 表示逻辑值 **true** 或 **false**。某些 MATLAB 函数和运算符返回逻辑值 **1** 或逻辑值 **0** 以指示某个条件是否为真。例如，语句 **(5 * 10) > 40** 返回逻辑值 **1**。

MATLAB char 数组

MATLAB **char** 数组将数据存储为无符号 16 位整数。要将 MATLAB **char** 数组转换为 C 样式的字符串，请调用 **mxArrayToString**。要将 C 样式的字符串转换为 **char** 数组，请调用 **mxCreateString**。

元胞数组

元胞数组是 MATLAB 数组的集合，其中每个 **mxArray** 称为一个元胞。元胞数组可以将不同类型的 MATLAB 数组存储在一起。元胞数组的存储方式与数值矩阵类似，区别在于数据部分包含一个由指向 **mxArrays** 的指针构成的向量。此向量的成员称为元胞。每个元胞可以是任何受支持的数据类型，甚至可以是另一个元胞数组。

结构体

1×**1** 结构体的存储方式与 **1**×**n** 元胞数组相同，其中 **n** 是结构体中的字段数。数据向量的成员称为字段。每个字段都与在 **mxArray** 中存储的一个名称关联。

对象

对象的存储和访问方式与结构体相同。在 MATLAB 中，对象是具有已注册方法的命名结构体。在 MATLAB 之外，作为一种结构体，对象包含用于标识对象名称的附加类名称的存储。

多维数组

任何类型的 MATLAB 数组都可以是多维的。整数向量的存储方式是每个元素都具有对应维度的大小。数据的存储方式与矩阵相同。

空数组

任何类型的 MATLAB 数组都可以是空数组。空 **mxArray** 表示它至少有一个维度等于零。例如，**double** 类型的双精度 **mxArray**，其中 **m** 和 **n** 等于 0 且 **pa** 是 **NULL**，则它是一个空数组。

稀疏矩阵

在 MATLAB 中，稀疏矩阵与满矩阵采用不同的存储约定。参数 **pa** 仍然是双精度数字或逻辑值的数组，但此数组仅包含非零数据元素。

还有三个参数：**nzmax**、**ir** 和 **jc**。声明这些参数的变量时，请使用 **mwSize** 和 **mwIndex** 类型。

- **nzmax** 是包含 **ir** 和 **pa** 长度的整数。它是稀疏矩阵中非零元素的最大数量。
- **ir** 指向长度 **nzmax** 的整数数组，其中包含 **pa** 中相应元素的行索引。
- **jc** 指向长度 **n+1** 的整数数组，其中 **n** 是稀疏矩阵中的列数。在 C 中，**mxArray** 的第一个元素的索引为 0。**jc** 数组包含列索引信息。如果稀疏矩阵的第 **j** 列具有任何非零元素，则 **jc[j]** 表示第 **j** 列中的第一个非零元素的 **ir** 和 **pa** 的索引。索引 **jc[j+1] - 1** 包含该列中的最后一个非零元素。对于稀疏矩阵的第 **j** 列，**jc[j]** 是前面所有列中非零元素的总数。**jc** 数组的最后一个元素 **jc[n]** 等于 **nnz**，即整个稀疏矩阵中非零元素的数量。如果 **nnz** 小于 **nzmax**，则可以在不分配更多存储的情况下将更多非零项插入数组中。

使用数据类型

您可以使用 C/C++ 编写源 MEX 文件、MAT 文件应用程序和引擎应用程序，它们接受 MATLAB 支持的任何类或数据类型（请参阅“数据类型”）。在 Fortran 中，仅支持创建双精度 **n×m** 数组和字符串。您可以使用二进制 C/C++ 和 Fortran MEX 文件，如 MATLAB 函数。

注意 MATLAB 不检查使用 Matrix Library 创建函数（例如 **mxCreateStructArray**）在 C/C++ 或 Fortran 中创建的 MATLAB 数据结构体的有效性。使用无效语法创建 MATLAB 数据结构体可能会导致您的 C/C++ 或 Fortran 程序出现意外行为。

声明数据结构体

要处理 MATLAB 数组，请使用 **mxArray** 类型。以下语句声明名为 **myData** 的 **mxArray**：

```
mxArray *myData;
```

要定义 **myData** 的值，请使用 **mxCreate*** 函数之一。一些数组创建例程很有用，如 **mxCreateNumericArray**、**mxCreateCellArray** 和 **mxCreateCharArray**。例如，以下语句分配一个初始化为 **0** 的 **m×1** 浮点 **mxArray**：

```
myData = mxCreateDoubleMatrix(m, 1, mxREAL);
```

C/C++ 编程人员应该注意，MATLAB 数组中的数据按列优先顺序排列。有关说明，请参阅“数据存储”（第 7-6 页）。使用 MATLAB **mxGet*** 数组访问例程来读取 **mxArray** 中的数据。

操作数据

mxGet* 数组访问例程获取对 **mxArray** 中数据的引用。使用这些例程来修改 MEX 文件中的数据。每个函数用于访问 **mxArray** 中的特定信息。这些实用的函数包括 **mxGetDoubles**、**mxGetComplexDoubles**、**mxGetM** 和 **mxGetString**。其中许多函数都有相应的 **mxSet*** 例程，您可用来修改数组中的值。

以下语句将输入 **prhs[0]** 读入 C 样式的字符串 **buf** 中。

```
char *buf;
int buflen;
```

```
int status;
buflen = mxGetN(prhs[0])*sizeof(mxChar)+1;
buf = mxMalloc(buflen);
status = mxGetString(prhs[0], buf, buflen);
```

explore 示例

MATLAB 附带一个名为 **explore.c** 的示例源 MEX 文件，用于标识输入变量的数据类型。此示例的源代码位于 **matlabroot/extern/examples/mex** 中，其中 **matlabroot** 表示在您的系统上安装 MATLAB 的顶层文件夹。

注意 在涉及文件夹路径的与平台无关的讨论中，本文档使用 UNIX® 约定。例如，对 **mex** 文件夹的一般引用是 **matlabroot/extern/examples/mex**。

要编译示例 MEX 文件，请首先将文件复制到路径上的一个可写文件夹中。

```
copyfile(fullfile(matlabroot,'extern','examples','mex','explore.c'),'.','f')
```

使用 **mex** 命令编译 MEX 文件。

```
mex explore.c -R2018a
```

键入：

```
x = 2;
explore(x)
```

```
------------------------------------------
Name: prhs[0]
Dimensions: 1x1
Class Name: double
------------------------------------------
  (1,1) = 2
```

explore 可以接受任何数据类型。尝试对以下示例使用 **explore**：

```
explore([1 2 3 4 5])
explore 1 2 3 4 5
explore({1 2 3 4 5})
explore(int8([1 2 3 4 5]))
explore {1 2 3 4 5}
explore(sparse(eye(5)))
explore(struct('name', 'Joe Jones', 'ext', 7332))
explore(1, 2, 3, 4, 5)
explore(complex(3,4))
```

另请参阅

详细信息

- “数据类型”

编译 C MEX 函数

此示例显示如何编译示例 C MEX 函数 **arrayProduct**。使用此示例验证系统的 C 编译配置。有关编译 C++ MEX 函数的信息，请参阅“编译 C++ MEX 程序” （第 9-8 页）。

验证您是否安装了支持的兼容编译器。有关安装编译器的信息，请访问供应商网站。

```
mex -setup C
```

将源 MEX 文件复制到您的路径上的可写文件夹中。

```
copyfile(fullfile(matlabroot,'extern','examples','mex','arrayProduct.c'),'.','f')
```

调用 **mex** 命令来编译该函数。

```
mex arrayProduct.c -R2018a
```

此命令将创建文件 **arrayProduct.ext**，其中 **ext** 是由 **mexext** 函数返回的值。

测试该函数。**arrayProduct** 函数接受一个标量并将其与 1×N 矩阵相乘。像调用 MATLAB 函数一样调用 **arrayProduct**。

```
s = 5;
A = [1.5, 2, 9];
B = arrayProduct(s,A)
```

```
B =
   7.5000   10.0000   45.0000
```

另请参阅

mex | **mexext**

详细信息

- “更改默认编译器” （第 7-11 页）
- “编译 C++ MEX 程序” （第 9-8 页）

外部网站

- 支持和兼容的编译器

更改默认编译器

更改 Windows 系统上的默认值

MATLAB 为 C、C++ 和 Fortran 语言文件维护单独的默认编译器选项。如果您的 Windows 系统上安装了某个语言的多个受 MATLAB 支持的编译器，则 MATLAB 会选择其中一个作为默认编译器。要更改默认值，请使用 **mex -setup lang** 命令。MATLAB 会显示一条消息，其中包含选择不同默认编译器的链接。

如果您调用 **mex -setup** 时未使用 **lang** 参数，则 MATLAB 会显示有关默认 C 编译器的信息。MATLAB 还会显示其他受支持语言的链接。要更改另一种语言的默认值，请选择一个链接。

如果通过操作系统提示符调用 **mex -setup**，MATLAB 会显示相同的信息。但这些消息不含链接。在这种情况下，MATLAB 显示用于更改默认编译器的相应 **mex** 命令语法。复制命令并将其粘贴到操作系统提示符下。

您所选的编译器将保留为该语言的默认值，直至您调用 **mex -setup** 选择其他默认值。

C 编译器

要更改默认的 C 编译器，请在 MATLAB 命令提示符下键入：

```
mex -setup
```

mex -setup 默认显示关于 C 编译器的信息。MATLAB 还显示系统上其他 C 编译器的链接。要更改默认值，请选择以下链接之一。

或者键入：

```
mex -setup c
```

C++ 编译器

要更改默认的 C++ 编译器，请键入：

```
mex -setup cpp
```

MATLAB 显示关于默认 C++ 编译器的信息，并提供系统上其他 C++ 编译器的链接。要更改默认值，请选择以下链接之一。有关示例，请参阅“选择 C++ 编译器” （第 8-15 页）。

Fortran 编译器

要更改默认的 Fortran 编译器，请键入：

```
mex -setup Fortran
```

更改 Linux 系统上的默认值

有关在 Linux 平台上将 gcc/g++ 编译器更改为受支持版本的信息，请参阅“Change Default gcc Compiler on Linux System”。

更改 macOS 系统上的默认值

如果您的系统中安装了多个版本的 Xcode，MATLAB 会使用 **Xcode.app** 应用程序所定义的编译器。您可以通过 **Xcode.X.app** 使用该编译器，其中 **Xcode.X.app** 是用于保存之前安装的 Xcode 版本的名称。

启动 MATLAB 之前，通过终端键入：

```
xcode-select -switch /Applications/Xcode.X.app/Contents/Developer
```

要查看 MATLAB 所使用的 Xcode 版本，请在终端键入：

```
xcode-select -p
```

请勿使用 mex -f optionsfile 语法

在以后的版本中，将删除用于指定编译配置文件的 **mex** 命令的 **-f** 选项。请改用以下主题中介绍的工作流来指定编译器。

另请参阅

mex

详细信息

- “选择 C++ 编译器” (第 8-15 页)
- “Change Default gcc Compiler on Linux System”

外部网站

- 支持和兼容的编译器

调用 LAPACK 和 BLAS 函数

您可以使用 MEX 文件调用 LAPACK 或 BLAS 函数。要创建 MEX 文件，您需要具备 C/C++ 或 Fortran 编程经验以及软件资源（编译器和链接器）来编译一个可执行文件。了解如何使用 Fortran 子例程也很有帮助。MATLAB 在 **matlabroot/extern/lib** 中提供了 **mwlapack** 和 **mwblas**。为帮助您快速入门，**matlabroot/extern/examples/refbook** 中提供了源代码示例。

要调用 LAPACK 或 BLAS 函数，请执行下列操作：

1. 创建包含 **mexFunction** 入口例程的源 MEX 文件。
2. 确保您所用的编译器受平台支持。有关支持的编译器的最新列表，请参阅支持和兼容的编译器。
3. 使用 **mex** 命令加上分离式复矩阵编译标志 **-R2017b** 编译二进制 MEX 文件。
   - 将源文件链接到库 **mwlapack** 或 **mwblas**，或者同时链接到两者。
   - **mwlapack** 和 **mwblas** 库仅支持 64 位整数作为矩阵维度。请勿使用 **-compatibleArrayDims** 选项。
   - 要通过使用复数的函数编译 MEX 文件，请参阅“Pass Separate Complex Numbers to Fortran Functions”。
4. 有关 BLAS 或 LAPACK 函数的信息，请参阅 https://netlib.org/blas/ 或 https://netlib.org/lapack/。

使用 BLAS 函数编译 matrixMultiply MEX 函数

此示例说明如何使用 BLAS 库中的函数编译示例 MEX 文件 **matrixMultiply.c**。要使用此文件，请将其复制到一个本地文件夹。例如：

```
copyfile(fullfile(matlabroot,'extern','examples','refbook','matrixMultiply.c'),'.')
```

示例文件是只读文件。要修改示例，请通过键入以下命令确保该文件是可写的：

```
fileattrib('matrixMultiply.c','+w')
```

要编译 MEX 文件，请键入：

```
mex -v -R2017b matrixMultiply.c -lmwblas
```

要运行 MEX 文件，请键入：

```
A = [1 3 5; 2 4 7];
B = [-5 8 11; 3 9 21; 4 0 8];
X = matrixMultiply(A,B)
```

```
X =
   24   35  114
   30   52  162
```

保留修改后的输入值

许多 LAPACK 和 BLAS 函数会修改向其传递的参数的值。在将可被修改的参数传递给这些函数之前，最好先制作这些参数的副本。有关 MATLAB 如何处理 **mexFunction** 的参数的信息，请参阅“Managing Input and Output Parameters”。

matrixDivide 示例

此示例调用 LAPACK 函数 **dgesv**，该函数会修改其输入参数。此示例中的代码会制作 **prhs[0]** 和 **prhs[1]** 的副本，并将副本传递给 **dgesv** 以保留输入参数的内容。

要查看该示例，请在 MATLAB 编辑器中打开 **matrixDivide.c**。要创建 MEX 文件，请将源文件复制到一个可写文件夹中。

```
copyfile(fullfile(matlabroot,'extern','examples','refbook','matrixDivide.c'),'.')
```

要编译该文件，请键入：

```
mex -v -R2017b matrixDivide.c -lmwlapack
```

要测试，请键入：

```
A = [1 2; 3 4];
B = [5; 6];
X = matrixDivide(A,B)
```

```
X =
   -4.0000
    4.5000
```

将参数从 C/C++ 程序传递给 Fortran 函数

LAPACK 和 BLAS 函数是用 Fortran 编写的。C/C++ 和 Fortran 使用不同约定在函数之间传递参数。Fortran 函数按引用传递参数，而 C/C++ 函数按值传递参数。当按值传递参数时，传递的是值的副本。当按引用传递参数时，传递的是值的指针。引用也就是值的地址。

当您从 C/C++ 程序调用 Fortran 子例程（如来自 LAPACK 或 BLAS 的函数）时，请务必按引用传递参数。要按引用传递参数，请在参数前面使用“与”符号 (**&**)，除非该参数本身已是引用。例如，使用 **mxGetDoubles** 函数创建矩阵时，您可以创建对矩阵的引用，不需要在参数前使用 & 符号。

在以下代码段中，变量 **m**、**n**、**p**、**one** 和 **zero** 需要 **&** 字符使其变为引用。变量 **A**、**B**、**C** 和 **chn** 是指针，它们是引用。

```
/* pointers to input & output matrices*/
double *A, *B, *C;
/* matrix dimensions */
mwSignedIndex m,n,p;
/* other inputs to dgemm */
char *chn = "N";
double one = 1.0, zero = 0.0;

/* call BLAS function */
dgemm(chn, chn, &m, &n, &p, &one, A, &m, B, &p, &zero, C, &m);
```

matrixMultiply 示例

matrixMultiply.c 示例调用 **dgemm**，即按引用传递所有参数。要查看源代码，请在 MATLAB 编辑器中打开 **matrixMultiply.c**。要编译并运行此示例，请参阅“使用 BLAS 函数编译 matrixMultiply MEX 函数” （第 7-13 页）。

将参数从 Fortran 程序传递给 Fortran 函数

您可以从 Fortran MEX 文件中调用 LAPACK 和 BLAS 函数。以下示例使用两个矩阵，并通过调用 BLAS 例程 **dgemm** 将这两个矩阵相乘。要运行该示例，请将代码复制到编辑器中并将文件命名为 **calldgemm.F**。

```
#include "fintrf.h"

      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
      mwPointer plhs(*), prhs(*)
      integer nlhs, nrhs
      mwPointer mxcreatedoublematrix
      mwPointer mxgetpr
      mwPointer A, B, C
      mwSize mxgetm, mxgetn
      mwSignedIndex m, n, p
      mwSize numel
      double precision one, zero, ar, br
      character ch1, ch2

      ch1 = 'N'
      ch2 = 'N'
      one = 1.0
      zero = 0.0

      A = mxgetpr(prhs(1))
      B = mxgetpr(prhs(2))
      m = mxgetm(prhs(1))
      p = mxgetn(prhs(1))
      n = mxgetn(prhs(2))

      plhs(1) = mxcreatedoublematrix(m, n, 0.0)
      C = mxgetpr(plhs(1))
      numel = 1
      call mxcopyptrtoreal8(A, ar, numel)
      call mxcopyptrtoreal8(B, br, numel)

      call dgemm(ch1, ch2, m, n, p, one, %val(A), m,
     +           %val(B), p, zero, %val(C), m)

      return
      end
```

链接到 BLAS 库，其中包含 **dgemm** 函数。

```
mex -v -R2017b calldgemm.F -lmwblas
```

在 UNIX 系统上修改函数名称

在 UNIX 系统上调用 LAPACK 或 BLAS 函数时，在函数名称后面添加下划线字符。例如，要调用 **dgemm**，请使用：

```
dgemm_(arg1, arg2, ..., argn);
```

或者将以下行添加到您的源代码中：

```
#if !defined(_WIN32)
#define dgemm dgemm_
#endif
```

另请参阅

外部网站

- https://netlib.org/lapack/
- https://netlib.org/blas/

升级 MEX 文件以使用 64 位 API

mex 命令默认使用 **-largeArrayDims** 选项。本主题描述如何升级 MEX 文件以使用 64 位 API。

您可以通过调用带有 **-compatibleArrayDims** 选项的 **mex** 命令继续使用 32 位 API。有关使用此选项的详细信息，请参阅“What If I Do Not Upgrade?”。

要查看并更新 MEX 文件的源代码，请使用以下检查清单。

1. 在编辑之前准备代码 - 请参阅“备份文件并创建测试” （第 7-17 页）。
2. 以迭代方式更改和测试代码。

   在使用 64 位 API 编译 MEX 文件之前，请使用“更新变量” （第 7-17 页）重构您的现有代码。对于 Fortran，则使用“Upgrade Fortran MEX Files to use 64-bit API” 重构您的代码。

   每次更改后，请编译并测试您的代码：

   - 使用 32 位 API 进行编译。例如，要编译 **myMexFile.c**，请键入：

     ```
     mex -compatibleArrayDims myMexFile.c
     ```
   - 每次重构后进行测试 - 请参阅“在每个重构迭代后进行测试、调试并解决存在的差异” （第 7-19 页）。
3. 使用 64 位 API 进行编译。要编译 **myMexFile.c**，请键入：

   ```
   mex myMexFile.c
   ```
4. 消除故障和警告 - 请参阅“解决 -largeArrayDims 编译故障和警告” （第 7-19 页）。
5. 比较结果 - 请参阅“执行 64 位 MEX 文件并与 32 位版本的结果进行比较” （第 7-20 页）。
6. 检查内存 - 请参阅“使用大型数组进行试验” （第 7-20 页）。

以下过程使用 C/C++ 术语和示例代码。Fortran MEX 文件存在相同的问题，需要执行“Upgrade Fortran MEX Files to use 64-bit API”中所述的更多任务。

备份文件并创建测试

在修改代码之前，确认 MEX 文件可与 32 位 API 结合使用。至少构建预期的输入和输出列表，或创建完整的测试套件。使用这些测试，将结果与更新后的源代码进行比较。结果应该相同。

备份所有源、二进制和测试文件。

更新变量

要处理大型数组，请转换包含数组索引或大小的变量，以使用 **mwSize** 和 **mwIndex** 类型，而不是 32 位 **int** 类型。查看您的代码是否包含以下变量类型：

- 矩阵 API 函数直接使用的变量 - 请参阅“更新用于调用 64 位 API 函数的参数” （第 7-18 页）。
- 中间变量 - 请参阅“更新用于数组索引和大小的变量” （第 7-18 页）。
- 用作大小/索引值和用作 32 位整数的变量 - 请参阅“分析其他变量” （第 7-18 页）。

更新用于调用 64 位 API 函数的参数

识别代码中使用 **mwSize / mwIndex** 类型的 64 位 API 函数。有关函数列表，请参阅“Using the 64-Bit API”。搜索您用于调用函数的变量。检查函数参考文档的**语法**标题下面显示的函数签名。该签名确定了取 **mwSize / mwIndex** 值作为输入值或输出值的变量。更改您的变量以使用正确的类型。

例如，假设您的代码使用以下语句中所示的 **mxCreateDoubleMatrix** 函数：

```
int nrows,ncolumns;
...
y_out = mxCreateDoubleMatrix(nrows, ncolumns, mxREAL);
```

要查看函数签名，请键入：

```
doc mxCreateDoubleMatrix
```

签名为：

```
mxArray *mxCreateDoubleMatrix(mwSize m, mwSize n,
   mxComplexity ComplexFlag)
```

输入参数 **m** 和 **n** 的类型为 **mwSize**。如表中所示更改代码。

替换：	为：
`int nrows,ncolumns;`	`mwSize nrows,ncolumns;`

更新用于数组索引和大小的变量

如果您的代码使用中间变量来计算大小和索引值，请将 **mwSize / mwIndex** 用于这些变量。例如，以下代码将 **mxCreateDoubleMatrix** 的输入声明为 **mwSize** 类型：

```
mwSize nrows,ncolumns;   /* inputs to mxCreateDoubleMatrix */
int numDataPoints;
nrows = 3;
numDataPoints = nrows * 2;
ncolumns = numDataPoints + 1;
...
y_out = mxCreateDoubleMatrix(nrows, ncolumns, mxREAL);
```

该示例使用中间变量 numDataPoints（类型为 **int**）来计算 ncolumns 的值。如果您将 nrows 中的 64 位值复制到 32 位变量 numDataPoints，则结果值将被截断。您的 MEX 文件可能崩溃或产生错误的结果。如下表中所示，将类型 **mwSize** 用于 numDataPoints。

替换：	为：
`int numDataPoints;`	`mwSize numDataPoints;`

分析其他变量

您无需更改代码中的每个整数变量。例如，结构体和状态代码中的字段编号为 **int** 类型。不过，您需要识别用于多种用途的变量，必要时将其替换为多个变量。

以下示例将基于传感器的数量创建矩阵 myNumeric 和结构体 myStruct。该代码将一个变量 numSensors 同时用于数组大小和结构体中的字段数量。

```
mxArray *myNumeric, *myStruct;
int numSensors;
mwSize m, n;
char **fieldnames;
...
myNumeric = mxCreateDoubleMatrix(numSensors, n, mxREAL);
myStruct = mxCreateStructMatrix(m, n, numSensors, fieldnames);
```

mxCreateDoubleMatrix 和 **mxCreateStructMatrix** 的函数签名为：

```
mxArray *mxCreateDoubleMatrix(mwSize m, mwSize n,
   mxComplexity ComplexFlag)
mxArray *mxCreateStructMatrix(mwSize m, mwSize n,
   int nfields, const char **fieldnames);
```

对于 **mxCreateDoubleMatrix** 函数，您的代码将 numSensors 用于变量 m。m 的类型为 **mwSize**。对于 **mxCreateStructMatrix** 函数，您的代码将 numSensors 用于变量 nfields。nfields 的类型为 **int**。要处理这两个函数，请将 numSensors 替换为两个新变量，如下表所示。

替换：	**为：**

`int numSensors;`	`/* create 2 variables */`
	`/* of different types */`
	`mwSize numSensorSize;`
	`int numSensorFields;`

`myNumeric =`
`  mxCreateDoubleMatrix(`
`  numSensors,`
`  n, mxREAL);`
	`/* use mwSize variable */`
	`/* numSensorSize */`
	`myNumeric =`
	`  mxCreateDoubleMatrix(`
	`  numSensorSize,`
	`  n, mxREAL);`

`myStruct =`
`  mxCreateStructMatrix(`
`  m, n,`
`  numSensors,`
`  fieldnames);`
	`/* use int variable */`
	`/* numSensorFields */`
	`myStruct =`
	`  mxCreateStructMatrix(`
	`  m, n,`
	`  numSensorFields,`
	`  fieldnames);`

在每个重构迭代后进行测试、调试并解决存在的差异

要使用 32 位 API 编译 **myMexFile.c**，请键入：

```
mex -compatibleArrayDims myMexFile.c
```

使用您在此过程开始时创建的测试，将更新的 MEX 文件的结果与原始二进制文件进行比较。两个 MEX 文件应返回相同的结果。否则，请调试并解决存在的任何差异。此时解决差异的难度比使用 64 位 API 编译时解决差异的难度要小。

解决 -largeArrayDims 编译故障和警告

在查看并更新代码之后，使用大型数组处理 API 来编译 MEX 文件。要使用 64 位 API 编译 **myMexFile.c**，请键入：

```
mex myMexFile.c
```

由于 **mwSize / mwIndex** 类型为 MATLAB 类型，因此您的编译器有时将其称为 **size_t**、**unsigned_int64** 或其他类似的名称。

大多数编译问题都与 32 位和 64 位类型之间的类型不匹配有关。请参考 How do I update MEX-files to use the large array handling API (-largeArrayDims)? 中的步骤 5，了解特定编译器的常见编译问题及可能的解决方案。

执行 64 位 MEX 文件并与 32 位版本的结果进行比较

将使用 64 位 API 编译的 MEX 文件的运行结果与原始二进制文件的结果进行比较。如果存在任何差异或故障，请使用调试器调查原因。有关调试器功能的信息，请参阅您的编译器文档。

要确定在运行 MEX 文件时可能遇到的问题及可能的解决方案，请参考 How do I update MEX-files to use the large array handling API (-largeArrayDims)? 中的步骤 6。

在解决问题并升级您的 MEX 文件后，它将在使用大型数组处理 API 的同时复制原始代码的功能。

使用大型数组进行试验

如果您可以访问具有大内存量的计算机，则可以使用大型数组进行试验。一个包含 2^{32} 个元素的双精度浮点数数组（MATLAB 中的默认值）需要占用大约 32 GB 的内存。

有关演示大型数组用法的示例，请参阅“Handling Large mxArrays in C MEX Files” 中的 **arraySize.c** MEX 文件。

另请参阅

相关示例

- “Upgrade Fortran MEX Files to use 64-bit API”
- “Handling Large mxArrays in C MEX Files”

详细信息

- “What If I Do Not Upgrade?”
- “Using the 64-Bit API”

外部网站

- How do I update MEX-files to use the large array handling API (-largeArrayDims)?

无效的 MEX 文件错误

如果 MATLAB 找不到 MEX 文件引用的所有 **.dll** 文件，则无法加载该 MEX 文件。MATLAB 显示以下错误消息：

```
Invalid MEX-file mexfilename:
The specified module could not be found.
```

其中 **mexfilename** 是依存关系有错误的模块。此模块找不到它所依赖的库。要解决此错误，请查找所依赖库的名称，并确定它们是否存在于您的系统和系统路径中。要查找库依赖项，请执行以下操作：

- 在 Windows 系统上，从网站 https://www.dependencywalker.com 下载 Dependency Walker 实用工具。
- 在 Linux 系统上，使用：

  ```
  ldd -d libname.so
  ```
- 在 macOS 系统上，使用：

  ```
  otool -L libname.dylib
  ```

对于 MEX 文件在编译时所链接的 **.dll** 文件，这些 **.dll** 文件必须位于系统路径上或位于与 MEX 文件相同的文件夹中。

MEX 文件可能需要其他未链接到 MEX 文件的库。未能找到其中一个显式加载的库可能不会导致 MEX 文件无法加载，但会导致其无法正常工作。加载库的代码控制用于查找这些库的搜索路径。搜索路径可能不包括包含 MEX 文件的文件夹。有关正确的安装位置，请查阅库文档。

可能的故障原因包括：

- MATLAB 版本不兼容。有关详细信息，请参阅“MEX 版本兼容性” （第 7-23 页）。
- 缺少编译器运行时库。如果您的系统没有包含用于编译 MEX 文件的相同编译器，请参阅 Microsoft MSDN® 网站，了解关于 Visual C++® 可再发行软件包的信息。
- 缺少或不正确地安装了专用运行时库。请联系您的 MEX 文件或库供应商。

另请参阅

详细信息

- “MEX 版本兼容性” （第 7-23 页）

外部网站

- 如何确定我的 MEX 文件或独立应用程序需要哪些库?

运行您从其他人处接收的 MEX 文件

要调用 MEX 文件，请将该文件置于您的 MATLAB 路径中。然后键入文件的名称（无需键入文件扩展名）。

如果您有 MEX 文件的源代码，请参阅“编译 C MEX 函数” （第 7-10 页），了解关于创建可执行函数的信息。

如果在调用并非您创建的 MEX 文件时遇到运行时错误，请考虑以下因素：

- “MEX Platform Compatibility”
- “MEX 版本兼容性” （第 7-23 页）
- 在 Windows 平台上，安装用于创建 MEX 文件的 C++ 编译器运行时库。如果您的计算机上没有安装与编译 MEX 文件所用编译器相同的编译器，则需要执行此步骤。
- 如果 MEX 文件使用专用运行时库，则必须在您的系统上安装这些库。

如果您编写并编译了 MEX 文件，然后在同一 MATLAB 会话中执行此文件，则可按预期使用所有依赖的库。但如果您的 MEX 文件是从其他 MATLAB 用户那里收到的，则您可能不具备所有依赖的库。

MEX 文件是 MATLAB 解释器在您调用函数时加载并执行的动态链接子例程。动态链接意味着在您调用函数时，程序才会查找依赖的库。MEX 文件使用 MATLAB 运行时库和语言特定的库。MEX 文件也可能使用专用运行时库。这些库的代码未包括在 MEX 文件中；在运行 MEX 文件时，您的计算机上必须存在这些库。

有关库依赖项故障排除的信息，请参阅“无效的 MEX 文件错误” （第 7-21 页）。

有关 MATLAB 如何查找 MEX 文件的信息，请参阅 “MATLAB 可访问的文件和文件夹”。

MEX 版本兼容性

为了获得最佳效果，您的 MATLAB 版本必须与用于创建该 MEX 文件的版本相同。

MEX 文件使用 MATLAB 运行时库。在早期版本的 MATLAB 上创建的 MEX 文件通常可在 MATLAB 的更高版本上运行。如果 MEX 文件出现错误，请从源代码重新编译 MEX 文件。

在 MATLAB 的较新版本上创建的 MEX 文件有时可以在 MATLAB 的旧版本上运行，但我们不提供这一支持。

另请参阅

详细信息

- “MEX Platform Compatibility”

8

C/C++ MEX 文件

使用 C 矩阵 API 创建 C++ MEX 函数

注意 MATLAB 提供使用现代 C++ 语义和设计模式的 API，即 “MATLAB Data API for C++”。MathWorks 建议您使用此 API 创建 MEX 函数。有关详细信息，请参阅“编写可从 MATLAB（MEX 文件）调用的 C++ 函数”。

如果您的 MEX 函数必须在 MATLAB R2017b 或更早版本中运行，则您必须在 C++ 应用程序中使用 “C Matrix API” 函数。用 C 矩阵 API 编译的 MEX 函数支持所有 C++ 语言标准。本主题讨论创建和使用 MEX 文件时要考虑的特定 C++ 语言事项。

您可以在 C++ 应用程序中使用 MATLAB C 代码示例。有关示例，请参阅 “C++ 类示例” （第 8-2 页）中的 **mexcpp.cpp**，其中包含 C 和 C++ 语句。

创建 C++ 源文件

MATLAB C++ 源代码示例使用 **.cpp** 文件扩展名。扩展名 **.cpp** 是 C++ 编译器可识别的明确扩展名。其他可能的扩展名包括 **.C**、**.cc** 和 **.cxx**。

编译和链接

要编译 C++ MEX 文件，请键入：

```
mex filename.cpp
```

其中，**filename** 为您的 MATLAB 路径上源代码文件的名称。

运行 C++ MEX 文件的系统上的 MATLAB 版本必须与编译该文件所用的 MATLAB 版本相同。

类析构函数的内存注意事项

不要在 MEX 函数中所用类的 C++ 析构函数中使用 **mxFree** 或 **mxDestroyArray** 函数。如果 MEX 函数引发错误，则 MATLAB 将清理 MEX 文件变量（如“Automatic Cleanup of Temporary Arrays in MEX Files”中所述）。

如果发生的错误导致对象超出范围，MATLAB 将调用 C++ 析构函数。直接在析构函数中释放内存意味着 MATLAB 和析构函数均释放相同的内存，而这可能损坏内存。

使用 mexPrintf 打印到 MATLAB 命令行窗口

在 C++ MEX 文件中，使用 **cout** 或 C 语言 **printf** 函数无法按预期工作。改用 **mexPrintf** 函数。

C++ 类示例

MEX 文件 **mexcpp.cpp** 说明如何通过 C 语言 MEX 文件来使用 C++ 代码。该示例使用 C 矩阵 API 中的函数。它使用成员函数、构造函数、析构函数和 **iostream** include 文件。

该函数定义一个具有成员函数 **display** 和 **set_data** 以及变量 **v1** 和 **v2** 的类 **myData**。它构造 **myData** 类的一个对象 **d**，并显示 **v1** 和 **v2** 的初始化值。然后，它将 **v1** 和 **v2** 设置为您的输入并显示新值。最后，使用 **delete** 操作符清理该对象。

要编译此示例，请将文件复制到 MATLAB 路径，并在命令提示符下键入：

```
mex mexcpp.cpp
```

调用语法是 **mexcpp(num1, num2)**。

C++ 文件处理示例

mexatexit.cpp 示例说明了 C++ 文件处理功能。将其与使用 **mexAtExit** 函数的 C 代码示例 **mexatexit.c** 进行比较。

C++ 示例

C++ 示例使用 **fileresource** 类来处理文件打开和关闭函数。MEX 函数调用此类的析构函数，它关闭数据文件。在此示例中，在对数据文件执行操作时还会在屏幕上显示一条消息。但是，在本例中，执行的唯一 C 文件操作是写入操作，即 **fprintf**。

要编译 **mexatexit.cpp** MEX 文件，请将文件复制到 MATLAB 路径并键入：

```
mex mexatexit.cpp
```

键入：

```
z = 'for the C++ MEX-file';
mexatexit(x)
mexatexit(z)
clear mexatexit
```

```
Writing data to file.
Writing data to file.
```

显示 **matlab.data** 的内容。

```
type matlab.data
```

```
my input string
for the C++ MEX-file
```

C 示例

此 C 代码示例注册 **mexAtExit** 函数，用于在清除 MEX 文件时执行清理任务（关闭数据文件）。此示例在执行文件操作 **fopen**、**fprintf** 和 **fclose** 时使用 **mexPrintf** 在屏幕上显示一条消息。

要编译 **mexatexit.c** MEX 文件，请将文件复制到 MATLAB 路径并键入：

```
mex mexatexit.c
```

运行示例。

```
x = 'my input string';
mexatexit(x)
```

```
Opening file matlab.data.
Writing data to file.
```

清除 MEX 文件。

```
clear mexatexit
```

```
Closing file matlab.data.
```

显示 **matlab.data** 的内容。

```
type matlab.data
```

```
my input string
```

另请参阅

mexPrintf | **mexAtExit**

相关示例

- **mexcpp.cpp**
- **mexatexit.cpp**
- **mexatexit.c**

详细信息

- “编译 C MEX 函数” （第 7-10 页）
- “C Matrix API”
- “编写可从 MATLAB（MEX 文件）调用的 C++ 函数”
- “MATLAB Data API for C++”

创建 C 源 MEX 文件 arrayProduct.c

此示例说明如何编写 MEX 文件，以在 MATLAB 中使用在 “C Matrix API” 中定义的 MATLAB 数组调用 C 函数 **arrayProduct**。您可以在此处 （第 8-0 页）查看完整的源文件。

要使用此示例，您需要：

- 能够编写 C 或 C++ 源代码。您使用 MATLAB 编辑器创建这些文件。
- MATLAB 支持的编译器。有关支持的编译器的最新列表，请参阅支持和兼容的编译器网站。
- “C Matrix API” 和 C MEX API 中的函数。
- **mex** 编译脚本。

如果您要使用自己的 C 开发环境，请参阅“Custom Build with MEX Script Options”了解详细信息。

C 函数 arrayProduct

以下代码定义 **arrayProduct** 函数，它将 1×n 矩阵 **y** 与标量值 **x** 相乘，并以数组 **z** 形式返回结果。您可以在 C++ 应用程序中使用这些相同的 C 语句。

```
void arrayProduct(double x, double *y, double *z, int n)
{
  int i;

  for (i=0; i<n; i++) {
    z[i] = x * y[i];
  }
}
```

创建源文件

打开 MATLAB 编辑器，创建一个文件，并在 MEX 文件中记录以下信息。

```
/*
 * arrayProduct.c - example in MATLAB External Interfaces
 *
 * Multiplies an input scalar (multiplier)
 * times a 1xN matrix (inMatrix)
 * and outputs a 1xN matrix (outMatrix)
 *
 * The calling syntax is:
 *
 *       outMatrix = arrayProduct(multiplier, inMatrix)
 *
 * This is a MEX file for MATLAB.
*/
```

添加包含 MATLAB API 函数声明的 C/C++ 头文件 **mex.h**。

```
#include "mex.h"
```

将文件保存在您的 MATLAB 路径（例如 **c:\work**）中，并将其命名为 **arrayProduct.c**。您的 MEX 文件名称为 **arrayProduct**。

创建入口例程

每个 C 程序都有一个 **main()** 函数。MATLAB 使用入口例程 **mexFunction** 作为函数的入口函数。添加以下 **mexFunction** 代码。

```
/* The gateway function */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
/* variable declarations here */

/* code here */
}
```

下表描述了 **mexFunction** 的输入参数。

参数	描述
nlhs	输出（左侧）参数的数量，或 **plhs** 数组的大小。
plhs	输出参数的数组。
nrhs	输入（右侧）参数的数量，或 **prhs** 数组的大小。
prhs	输入参数的数组。

验证 MEX 文件的输入和输出参数

使用 **nrhs** 和 **nlhs** 参数验证 MEX 文件输入和输出参数的数量。

要检查两个输入参数 **multiplier** 和 **inMatrix**，请使用以下代码。

```
if(nrhs != 2) {
   mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nrhs",
                     "Two inputs required.");
}
```

使用以下代码检查一个输出参数，即乘积 **outMatrix**。

```
if(nlhs != 1) {
   mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nlhs",
                     "One output required.");
}
```

使用 **plhs** 和 **prhs** 参数验证参数类型。以下代码将验证 **multiplier**（由 **prhs[0]** 表示）为标量。

```
/* make sure the first input argument is scalar */
if( !mxIsDouble(prhs[0]) ||
    mxIsComplex(prhs[0]) ||
    mxGetNumberOfElements(prhs[0]) != 1 ) {
   mexErrMsgIdAndTxt("MyToolbox:arrayProduct:notScalar",
                     "Input multiplier must be a scalar.");
}
```

以下代码将验证 **inMatrix**（由 **prhs[1]** 表示）为 **double** 类型。

```
if( !mxIsDouble(prhs[1]) ||
    mxIsComplex(prhs[1])) {
   mexErrMsgIdAndTxt("MyToolbox:arrayProduct:notDouble",
```

```
      "Input matrix must be type double.");
}
```

验证 **inMatrix** 为行向量。

```
/* check that number of rows in second input argument is 1 */
if(mxGetM(prhs[1]) != 1) {
   mexErrMsgIdAndTxt("MyToolbox:arrayProduct:notRowVector",
                     "Input must be a row vector.");
}
```

创建计算例程

添加 **arrayProduct** 代码。以下函数是您的计算例程，即用于执行您希望在 MATLAB 中使用的功能的源代码。

```
void arrayProduct(double x, double *y, double *z, int n)
{
  int i;

  for (i=0; i<n; i++) {
    z[i] = x * y[i];
  }
}
```

计算例程是可选的。您也可以将代码置于 **mexFunction** 函数块中。

编写代码以实现跨平台的灵活性

MATLAB 提供了基于平台表示整数大小值的预处理器宏 **mwsize**。计算例程将数组的大小声明为 **int**。将变量 **n** 和 **i** 的 **int** 声明替换为 **mwsize**。

```
void arrayProduct(double x, double *y, double *z, mwSize n)
{
  mwSize i;

  for (i=0; i<n; i++) {
    z[i] = x * y[i];
  }
}
```

声明计算例程的变量

将以下变量声明置于 **mexFunction** 中。

- 声明输入参数的变量。

  ```
  double multiplier;     /* input scalar */
  double *inMatrix;      /* 1xN input matrix */
  ```

- 为输入矩阵的大小声明 **ncols**。

  ```
  mwSize ncols;          /* size of matrix */
  ```

- 声明输出参数 **outMatrix**。

  ```
  double *outMatrix;     /* output matrix */
  ```

稍后，将 **mexFunction** 参数赋给这些变量。

读取输入数据

要读取标量输入，请使用 **mxGetScalar** 函数。

```
/* get the value of the scalar input  */
multiplier = mxGetScalar(prhs[0]);
```

使用 **mxGetDoubles** 函数指向输入矩阵数据。

```
/* create a pointer to the real data in the input matrix  */
inMatrix = mxGetDoubles(prhs[1]);
```

使用 **mxGetN** 函数获取矩阵大小。

```
/* get dimensions of the input matrix */
ncols = mxGetN(prhs[1]);
```

准备输出数据

要创建输出参数 **plhs[0]**，请使用 **mxCreateDoubleMatrix** 函数。

```
/* create the output matrix */
plhs[0] = mxCreateDoubleMatrix(1,ncols,mxREAL);
```

使用 **mxGetDoubles** 函数将 **outMatrix** 参数赋给 **plhs[0]**

```
/* get a pointer to the real data in the output matrix */
outMatrix = mxGetDoubles(plhs[0]);
```

执行计算

将参数传递给 **arrayProduct**。

```
/* call the computational routine */
arrayProduct(multiplier,inMatrix,outMatrix,ncols);
```

查看完整的源文件

将源文件与位于 **matlabroot/extern/examples/mex** 的 **arrayProduct.c** 进行比较。在编辑器中打开文件 **arrayProduct.c**。

有关使用 “MATLAB Data API for C++” 的 C++ MEX 文件示例，请参阅 **arrayProduct.cpp**。有关使用此 API 创建 MEX 文件的信息，请参阅 “C++ MEX 函数” （第 9-2 页）。

编译 MEX 函数

在 MATLAB 命令提示符下，使用 **mex** 命令编译该函数。

```
mex arrayProduct.c -R2018a
```

测试 MEX 函数

```
s = 5;
A = [1.5, 2, 9];
B = arrayProduct(s,A)
```

```
B =
   7.5000   10.0000   45.0000
```

验证 MEX 文件输入参数

最好在调用 MEX 函数之前验证 MATLAB 变量的类型。要测试输入变量 **inputArg** 并在必要时将其转换为 **double**，请使用以下代码。

```
s = 5;
A = [1.5, 2, 9];
inputArg = int16(A);
if ~strcmp(class(inputArg),'double')
   inputArg = double(inputArg);
end
B = arrayProduct(s,inputArg)
```

另请参阅

mex | **mexFunction** | **mxCreateDoubleMatrix**

相关示例

- **arrayProduct.c**
- “C Matrix API”
- “使用 C 矩阵 API 创建 C++ MEX 函数” （第 8-2 页）
- **arrayProduct.cpp**
- “MATLAB Data API for C++”
- “C++ MEX 函数” （第 9-2 页）

MEX 函数源代码示例表

下列各表包含用于创建示例 MEX 函数的源代码文件的列表。您可将这些示例作为起点来创建您自己的 MEX 函数。这些表包含以下信息。

- 示例名称 - 一个链接，您可以通过该链接直接在 MATLAB 编辑器中打开示例源文件。您可以使用任何代码开发编辑器来创建源 MEX 文件。
- 示例子文件夹 - **matlabroot/extern/examples** 中包含示例的子文件夹。将文件复制到可写文件夹时使用此子文件夹名称。
- 描述 - 描述该示例。
- 详细信息 - 指向有关描述或使用该示例的主题的链接，或指向示例中使用的 API 函数的链接。

快速入门

使用 **mex** 命令编译示例。请确保您安装了 MATLAB 支持的编译器。要验证为源代码语言 **lang** 选择的编译器，请键入：

```
mex -setup lang
```

使用以下命令语法将文件复制到路径上的一个可写文件夹中。**filename** 是示例的名称，**foldername** 是子文件夹名称。

```
copyfile(fullfile(matlabroot,'extern','examples','foldername','filename'),'.','f')
```

例如，要复制 **arrayProduct.c**，请键入：

```
copyfile(fullfile(matlabroot,'extern','examples','mex','arrayProduct.c'),'.','f')
```

C、C++ 和 Fortran MEX 函数

要在 MATLAB 中或在您的操作系统提示符下编译示例 MEX 函数，请使用以下命令语法。**filename** 是示例名称，**release-option** 指定示例使用的 API。有关 MATLAB API 的信息，请参阅 “选择 MEX 应用程序” （第 7-3 页）。

```
mex -v -release-option filename
```

示例名称	示例子文件夹	描述	更多信息
arrayFillGetPr.c	**refbook**	使用 **mxGetDoubles** 填充 **mxArray**。	“Fill mxArray in C MEX File”
arrayFillSetData.c	**refbook**	使用非双精度值填充 **mxArray**。	“Fill mxArray in C MEX File”
arrayFillSetPr.c	**refbook**	使用 **mxSetDoubles** 填充 **mxArray** 以动态分配内存。	“Fill mxArray in C MEX File”
arrayFillSetComplexPr.c	**refbook**	使用 **mxSetComplexDoubles** 填充 **mxArray** 以动态分配内存。	“Fill mxArray in C MEX File”
arrayProduct.c	**mex**	将标量乘以 1xN 矩阵。	“创建 C 源 MEX 文件 arrayProduct.c” （第 8-5 页）
arrayProduct.cpp	**cpp_mex**	与 **arrayProduct.c** 相同，使用 “MATLAB Data API for C++” 。	“C++ MEX 函数” （第 9-2 页）

示例名称	示例子文件夹	描述	更多信息
arraySize.c	**mex**	说明大型 **mxArray** 的内存要求。	“Handling Large mxArrays in C MEX Files”
complexAdd.F	**refbook**	添加两个复数双精度数组。	
convec.c convec.F	**refbook**	传递复数数据。	“Handle Complex Data in C MEX File”
dblmat.F compute.F	**refbook**	使用 Fortran **%VAL**。	
doubleelement.c	**refbook**	使用无符号 16 位整数。	“Handle 8-, 16-, 32-, and 64-Bit Data in C MEX File”
explore.c	**mex**	识别输入变量的数据类型。	“Work with mxArrays”
findnz.c	**refbook**	使用 N 维数组。	“Manipulate Multidimensional Numerical Arrays in C MEX Files”
fulltosparseIC.c fulltosparse.c fulltosparse.F, loadsparse.F	**refbook**	填充稀疏矩阵。	“Handle Sparse Arrays in C MEX File”
matsq.F	**refbook**	在 Fortran 中传递矩阵。	
matsqint8.F	**refbook**	在 Fortran 中传递非双精度矩阵。	
mexatexit.c mexatexit.cpp	**mex**	注册 exit 函数以关闭数据文件。	“C++ 文件处理示例” （第 8-3 页）
mexcallmatlab.c	**mex**	调用内置的 MATLAB **disp** 函数。	
mexcallmatlabwithtrap.c	**mex**	如何捕获错误信息。	
mexcpp.cpp	**mex**	在使用 C 矩阵 API 编译的 MEX 文件中演示一些 C++ 语言功能。	“C++ 类示例” （第 8-2 页）
mexevalstring.c	**mex**	使用 **mexEvalString** 分配 MATLAB 中的变量。	**mexEvalString**
mexfunction.c	**mex**	如何使用 **mexFunction**。	**mexFunction**
mxgetproperty.c	**mex**	使用 **mxGetProperty** 和 **mxSetProperty** 来更改图形对象的 **Color** 属性。	**mxGetProperty** 和 **mxSetProperty**
mexgetarray.c	**mex**	使用 **mexGetVariable** 和 **mexPutVariable** 来跟踪 MEX 文件和 MATLAB 全局工作区中的计数器。	**mexGetVariable** 和 **mexPutVariable**
mexgetarray.cpp		与 **mexgetarray.c** 相同，使用 “MATLAB Data API for C++” 中的 “getVariable” 和 “setVariable”。	“Set and Get MATLAB Variables from MEX”

示例名称	示例子文件夹	描述	更多信息
mexlock.c mexlockf.F	**mex**	如何锁定和解锁 MEX 文件。	**mexLock**
mxcalcsinglesubscript.c	**mx**	演示 MATLAB 从 1 开始的矩阵索引与从 0 开始的 C 索引之间的对比。	**mxCalcSingleSubscript**
mxcreatecellmatrix.c mxcreatecellmatrixf.F	**mx**	创建二维元胞数组。	“Create 2-D Cell Array in C MEX File”
mxcreatecharmatrixfromstr.c	**mx**	创建二维字符数组。	**mxCreateCharMatrixFromStrings**
mxcreatestructarray.c	**mx**	基于 C 结构体创建 MATLAB 结构体。	**mxCreateStructArray**
mxcreateuninitnumericmatrix.c	**mx**	创建未初始化的 **mxArray**，用本地数据进行填充，然后返回。	**mxCreateUninitNumericMatrix**
mxgeteps.c mxgetepsf.F	**mx**	读取 MATLAB **eps** 值。	**mxGetEps**
mxgetinf.c	**mx**	读取 **inf** 值。	**mxGetInf**
mxgetnzmax.c	**mx**	显示稀疏矩阵中的非零元素数量以及它能存储的最大非零元素数量。	**mxGetNzmax**
mxisclass.c	**mx**	检查数组是否为指定类的成员。	**mxIsClass**
mxisfinite.c	**mx**	检查 **NaN** 和无限值。	**mxIsFinite**
mxislogical.c	**mx**	检查工作区变量是逻辑值还是全局变量。	**mxIsLogical**
mxisscalar.c	**mx**	检查输入变量是否为标量。	**mxIsScalar**
mxmalloc.c	**mx**	分配内存，以将 MATLAB **char** 数组复制到 C 样式的字符串。	**mxMalloc**
mxsetdimensions.c mxsetdimensionsf.F	**mx**	重构数组。	**mxSetDimensions**
mxsetnzmax.c	**mx**	为稀疏矩阵重新分配内存并重置 **pr**、**pi**、**ir** 和 **nzmax** 的值。	**mxSetNzmax**
passstr.F	**refbook**	将 C 字符矩阵从 Fortran 传递到 MATLAB。	
phonebook.c	**refbook**	操作结构体和元胞数组。	“Pass Structures and Cell Arrays in C MEX File”
phonebook.cpp	**cpp_mex**	与 **phonebook.c** 相同，使用“MATLAB Data API for C++”。	“C++ MEX 函数” (第 9-2 页)
revord.c revord.F	**refbook**	将 MATLAB **char** 数组复制为 C 样式的字符串，或进行反向操作。	“Pass Strings in C MEX File”

示例名称	示例子文件夹	描述	更多信息
sincall.c sincall.F, fill.F	**refbook**	创建 **mxArray** 并将其传递给 MATLAB **sin** 和 **plot** 函数。	
timestwo.c timestwo.F	**refbook**	演示 MEX 文件的常见工作流。	“Pass Scalar Values in C MEX File”
xtimesy.c xtimesy.F	**refbook**	传递多个参数。	
yprime.c yprimef.F, yprimefg.F	**mex**	求解简单的三体轨道问题。	
yprime.cpp	**cpp_mex**	与 **yprime.c** 相同，使用“MATLAB Data API for C++”。	“C++ MEX 函数” （第 9-2 页）

MEX 函数调用 Fortran 子例程

此中的示例从 MEX 函数调用 LAPACK 或 BLAS 函数。这些示例链接到 Fortran 库 **mwlapack** 或 **mwblas** 或者同时链接到这两个库。要编译 MEX 函数，请按照**详细信息**列中列出的主题中的说明进行操作。

示例名称	示例子文件夹	描述	更多信息
dotProductComplexIC.c dotProductComplexIC.F dotProductComplex.c	**refbook**	为通过 C 或 Fortran MEX 文件调用的函数处理 Fortran 复数返回类型。**dotProductComplexIC.c** 和 **dotProductComplexIC.F** 使用交错式复矩阵 API。**dotProductComplex.c** 使用分离式复矩阵 API。	“Handle Fortran Complex Return Type — dotProductComplex”
matrixDivide.c	**refbook**	调用 LAPACK 函数。	“保留修改后的输入值” （第 7-13 页）
matrixDivideComplex.c	**refbook**	使用复数调用 LAPACK 函数。	“Pass Complex Variables — matrixDivideComplex”
matrixMultiply.c	**refbook**	调用 BLAS 函数。	“将参数从 C/C++ 程序传递给 Fortran 函数” （第 7-14 页）
utdu_slv.c	**refbook**	将 LAPACK 用于对称不定因式分解。	“Symmetric Indefinite Factorization Using LAPACK — utdu_slv”

另请参阅

mex

详细信息

- “选择 MEX 应用程序” （第 7-3 页）

- 支持和兼容的编译器
- “编译 C MEX 函数” (第 7-10 页)
- “更改默认编译器” (第 7-11 页)
- “MATLAB Support for Interleaved Complex API in MEX Functions”

选择 C++ 编译器

MATLAB 选择一个默认编译器，用于编译 MEX 文件，创建 MATLAB 的 C++ 库接口，以及独立的 MATLAB 引擎和 MAT 文件应用程序。C++ 应用程序的默认编译器可能不同于 C 应用程序的默认编译器。要查看默认 C++ 编译器，请键入以下命令之一：

```
mex -setup cpp
mex -setup CPP
mex -setup c++
```

MATLAB 还显示系统上其他 C++ 编译器的链接（如果有）。要更改默认值，请选择以下链接之一。

当您键入以下命令时，MATLAB 仅显示默认 C 编译器的信息。

```
mex -setup
```

选择 Microsoft Visual Studio 编译器

此示例说明在您的系统中有多个版本的 Microsoft Visual Studio® 时，如何确定和更改用于编译 C++ 应用程序的默认编译器。

要显示在您的系统上安装的 C++ 编译器的信息，请键入：

```
mex -setup cpp
```

要更改默认值，请点击其中一个链接。MATLAB 会显示该编译器的信息，在您调用 **mex -setup cpp** 选择不同默认值之前，该编译器为默认编译器。

选择 MinGW -w64 编译器

如果在您的系统中仅安装了 MinGW® 编译器，则 MATLAB 会自动为 C 和 C++ 应用程序选择 MinGW。如果您有多个 C 或 C++ 编译器，请键入以下命令以选择 C 编译器。

```
mex -setup
```

键入以下命令以选择 C++ 编译器。

```
mex -setup cpp
```

如果您仅键入 **mex -setup** 并选择 MinGW，则在编译 C++ 文件时，**mex** 可能会选择其他编译器。

另请参阅

mex | **clibgen.generateLibraryDefinition**

详细信息

- “更改默认编译器” （第 7-11 页）

提示用户在 C MEX 文件中提供输入

由于 MATLAB 不使用 **stdin** 和 **stdout**，因此不要使用 **scanf** 和 **printf** 等 C/C++ 函数来提示用户进行输入。以下示例说明如何使用 **mexCallMATLAB** 和 **input** 函数从用户获取数字。

另请参阅

mexCallMATLAB | **input** | **inputdlg**

相关示例

- “MEX 函数源代码示例表” (第 8-10 页)

在 Microsoft Windows 平台上调试

此示例说明调试 **yprime.c** 的一般步骤，可在您的 **matlabroot/extern/examples/mex/** 文件夹中找到该文件。有关使用 Visual Studio 的具体信息，请参考您的 Microsoft 文档。例如，请参阅 How can I debug a MEX file on Microsoft Windows Platforms with Microsoft Visual Studio 2017?

1. 确保 Visual Studio 是您选择的 C 编译器：

   ```
   cc = mex.getCompilerConfigurations('C','Selected');
   cc.Name
   ```

2. 使用 **-g** 选项编译源 MEX 文件，此选项会在编译文件时包含调试符号。例如：

   ```
   copyfile(fullfile(matlabroot,'extern','examples','mex','yprime.c'),'.','f')
   mex -g yprime.c
   ```

3. 启动 Visual Studio。不要退出您的 MATLAB 会话。
4. 有关附加 MATLAB 进程的信息，请参考您的 Visual Studio 文档。
5. 有关在代码中设置断点的信息，请参考 Visual Studio 文档。
6. 打开 MATLAB，然后键入：

   ```
   yprime(1,1:4)
   ```

   yprime.c 将在 Visual Studio 调试器中打开并停在第一个断点处。
7. 如果您选择**调试** > **继续**，MATLAB 将显示：

   ```
   ans =

     2.0000   8.9685   4.0000  -1.0947
   ```

另请参阅

详细信息

- How to debug MEX-files in Eclipse (Mars) compiled with MinGW64 and the -g flag in MATLAB R2017b and newer
- How can I debug a MEX file on Microsoft Windows Platforms with Microsoft Visual Studio 2017?
- “Debug in Simulink Environment” (Simulink)
- “MATLAB 代码分析” (MATLAB Coder)

C MEX 文件中的类型化数据访问

C 和 Fortran 矩阵 API 中的函数 **mxGetPr** 和 **mxGetPi** 可以读取 **mxArrays** 中 **mxDOUBLE_CLASS** 类型的数据元素。但是，这两个函数不会验证输入参数的数组类型。要进行类型安全的数据访问，请使用 C 语言的 **mxGetDoubles** 和 **mxGetComplexDoubles** 函数或者 Fortran 语言的 **mxGetDoubles** 和 **mxGetComplexDoubles** 函数。每个数值 **mxArray** 类型都有类型化的数据访问函数，如下表所示。

类型化数据访问函数是 C 和 Fortran 交错式复矩阵 API 的一部分，可以使用 **mex -R2018a** 选项来编译 MEX 函数。

MATLAB mxArray 类型	C 类型化数据访问函数	Fortran 类型化数据访问函数
mxDOUBLE_CLASS	mxGetDoubles mxSetDoubles mxGetComplexDoubles mxSetComplexDoubles	mxGetDoubles mxSetDoubles mxGetComplexDoubles mxSetComplexDoubles
mxSINGLE_CLASS	mxGetSingles mxSetSingles mxGetComplexSingles mxSetComplexSingles	mxGetSingles mxSetSingles mxGetComplexSingles mxSetComplexSingles
mxINT8_CLASS	mxGetInt8s mxSetInt8s mxGetComplexInt8s mxSetComplexInt8s	mxGetInt8s mxSetInt8s mxGetComplexInt8s mxSetComplexInt8s
mxUINT8_CLASS	mxGetUint8s mxSetUint8s mxGetComplexUint8s mxSetComplexUint8s	mxGetUint8s mxSetUint8s mxGetComplexUint8s mxSetComplexUint8s
mxINT16_CLASS	mxGetInt16s mxSetInt16s mxGetComplexInt16s mxSetComplexInt16s	mxGetInt16s mxSetInt16s mxGetComplexInt16s mxSetComplexInt16s
mxUINT16_CLASS	mxGetUint16s mxSetUint16s mxGetComplexUint16s mxSetComplexUint16s	mxGetUint16s mxSetUint16s mxGetComplexUint16s mxSetComplexUint16s
mxINT32_CLASS	mxGetInt32s mxSetInt32s mxGetComplexInt32s mxSetComplexInt32s	mxGetInt32s mxSetInt32s mxGetComplexInt32s mxSetComplexInt32s
mxUINT32_CLASS	mxGetUint32s mxSetUint32s mxGetComplexUint32s mxSetComplexUint32s	mxGetUint32s mxSetUint32s mxGetComplexUint32s mxSetComplexUint32s
mxINT64_CLASS	mxGetInt64s mxSetInt64s mxGetComplexInt64s mxSetComplexInt64s	mxGetInt64s mxSetInt64s mxGetComplexInt64s mxSetComplexInt64s

MATLAB mxArray 类型	C 类型化数据访问函数	Fortran 类型化数据访问函数
mxUINT64_CLASS	mxGetUint64s mxSetUint64s mxGetComplexUint64s mxSetComplexUint64s	mxGetUint64s mxSetUint64s mxGetComplexUint64s mxSetComplexUint64s

另请参阅

详细信息

- explore.c
- complexAdd.F
- “MEX 函数源代码示例表” (第 8-10 页)

MinGW -w64 编译器

您可以使用 MinGW-w64 编译器来编译 MEX 文件、MATLAB 的 C++ 库接口以及独立的 MATLAB 引擎和 MAT 文件应用程序。有关详细信息，请参阅“MATLAB 对 MinGW-w64 C/C++ 编译器的支持”。

安装 MinGW-w64 编译器

要安装该编译器，请使用“附加功能”菜单。

- 在 MATLAB **主页**选项卡的**环境**部分，点击**附加功能 > 获取附加功能**。
- 搜索 MinGW 或从**功能**菜单中选择。

编译 yprime.c 示例

您可以通过编译 **yprime.c** 示例来测试 MinGW 编译器。将源文件复制到可写文件夹。

```
copyfile(fullfile(matlabroot,'extern','examples','mex','yprime.c'),'.','f')
```

如果您在系统中仅安装了 MinGW 编译器，则 **mex** 命令会自动选择 MinGW。继续下一步。但如果您有多个 C 或 C++ 编译器，请使用 **mex -setup** 选择 MinGW。

```
mex -setup
```

编译该 MEX 文件。

```
mex yprime.c
```

MATLAB 将显示“编译工具”消息，其中显示了用于编译 MEX 文件的编译器。

运行函数。

```
yprime(1,1:4)
```

有关详细信息，请参阅“使用 MinGW -w64 编译 C/C++ MEX 文件的限制和疑难解答”（第 8-22 页）。

MinGW 安装文件夹名称不能包含空格

不要将 MinGW 安装在路径名中包含空格的位置。例如，不要使用：

```
C:\Program Files\mingw-64
```

应改用：

```
C:\mingw-64
```

更新 MEX 文件以使用 MinGW 编译器

如果您有使用 MATLAB 支持的其他编译器编译的 MEX 源文件，您可能需要修改代码，才能使用 MinGW 编译器进行编译。例如：

- Microsoft Visual Studio 所生成的库 (**.lib**) 文件与 MinGW 不兼容。

- 当使用 C++ MEX 文件中的 **mexErrMsgIdAndTxt** 函数引发异常时将无法进行对象清理，从而导致内存泄漏。
- 使用 MinGW 编译的 C++ MEX 文件中出现的未捕获异常会导致 MATLAB 崩溃。
- 不能编译其变量包含大量数据的 MEX 文件，因为编译器会耗尽内存。

另请参阅

详细信息

- “附加功能”
- “获取和管理附加功能”
- “使用 MinGW -w64 编译 C/C++ MEX 文件的限制和疑难解答” (第 8-22 页)
- 支持和兼容的编译器
- https://www.mathworks.com/matlabcentral/fileexchange/52848-matlab-support-for-mingw-w64-c-c-compiler

使用 MinGW -w64 编译 C/C++ MEX 文件的限制和疑难解答

不要链接到使用非 MinGW 编译器编译的库文件

如果您使用 MinGW 编译器编译的 MEX 文件链接到使用非 MinGW 编译器（例如 Microsoft Visual Studio）编译的库，该文件将不会在 MATLAB 中运行。由不同编译器生成的库 (**.lib**) 文件彼此不兼容。

您可以使用来自 MinGW 的 **dlltool** 实用工具来生成新的库文件。

MinGW 安装文件夹名称不能包含空格

不要将 MinGW 安装在路径名中包含空格的位置。例如，不要使用：

```
C:\Program Files\mingw-64
```

应改用：

```
C:\mingw-64
```

MEX 命令不选择 MinGW

如果在您的系统中仅安装了 MinGW 编译器，则 **mex** 命令会自动为 C 和 C++ MEX 文件选择 MinGW。如果您有多个 C 或 C++ 编译器，请使用 **mex -setup** 来为 C 和 C++（如果需要）MEX 文件选择 MinGW。

```
mex -setup
mex -setup cpp
```

如果您仅键入了 **mex -setup** 来选择 MinGW，则在编译 C++ 文件时，**mex** 可能会选择其他编译器。

手动为 MATLAB 配置 MinGW

从 MATLAB **附加功能**菜单安装 MinGW 时，MATLAB 会自动检测 MinGW 编译器。

如果需要，您可以手动配置 MinGW；如果您有 Windows 管理特权，可以使用 **configuremingw** 脚本。要下载此脚本，请参阅 MATLAB Answers 文章 "I already have MinGW on my computer.How do I configure it to work with MATLAB"。

MinGW 的行为与 Linux 中的 gcc/g++ 相似

当使用 **mex** 命令修改编译器标识符时，请使用 Linux 编译器标识符 **CFLAGS** 或 **CXXFLAGS**，而不是 Windows 标识符 **COMPFLAGS**。

C++ MEX 文件内有关使用 MEX 异常的潜在内存泄漏

使用 MinGW-w64 编译器编译的 C++ MEX 文件中的错误处理与 MATLAB 的错误处理不一致。如果 C++ MEX 文件包含某个类，则使用 **mexErrMsgIdAndTxt** 函数引发 MEX 异常可能导致为该类创建的对象发生内存泄漏。

MathWorks 建议您使用 C++ MEX API，而不是 C 矩阵 API。有关详细信息，请参阅“编写可从 MATLAB（MEX 文件）调用的 C++ 函数”。

例如，以下 C++ MEX 函数包含类 **MyClass**。

```
#include "mex.h"

class MyClass {
   public:
      MyClass() {
         mexPrintf("Constructor called");
      }
      ~MyClass() {
         mexPrintf("Destructor called");
      }
};

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
   MyClass X;

   if (nrhs != 0) {
      mexErrMsgIdAndTxt("MATLAB:cppfeature:invalidNumInputs",
            "No input arguments allowed.");
   }
}
```

该 MEX 函数从 **MyClass** 创建了对象 **X**，然后检查输入参数的数量。如果该 MEX 函数调用 **mexErrMsgIdAndTxt**，则 MATLAB 错误处理不会释放对象 **X** 的内存，从而产生内存泄漏。

C++ MEX 文件中未处理的显式异常意外终止 MATLAB

如果 C++ MEX 文件中的某个函数引发一个显式异常，而该异常未在 MEX 文件内使用 **catch** 语句捕获，则该异常会导致 MATLAB 终止，而不是向 MATLAB 命令行传播错误。

```
#include "mex.h"

class error {}; // Throw an exception of this class

class MyClass
{
   public:
   MyClass(){
      mexPrintf("Constructor called.");
   }
   ~MyClass(){
      mexPrintf("Destructor called.");
   }
};

void doErrorChecking(const MyClass& obj)
{
   // Do error checking
   throw error();
}

void createMyClass()
```

```
{
   MyClass myobj;
   doErrorChecking(myobj);
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
   createMyClass();
}
```

MEX 函数调用 **createMyClass**，后者创建 **MyClass** 类的一个对象，并调用函数 **doErrorChecking**。函数 **doErrorChecking** 会引发 **error** 类型的异常。但此异常未在 MEX 文件内捕获，从而导致 MATLAB 崩溃。

对于从 **std::exception** 类继承的类，也会发生此行为。

解决方法

在 MEX 函数中捕获异常：

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
   try{
      createMyClass();
   }
   catch(error e){
      // Error handling
   }
}
```

另请参阅

mexErrMsgIdAndTxt

详细信息

- “MinGW -w64 编译器” (第 8-20 页)

9

C++ MEX 应用程序

C++ MEX 函数

MEX（即 MEX 可执行程序）指自动加载的、可以像任何 MATLAB 函数一样调用的程序。

C++ MEX API

C++ MEX 函数基于两个 C++ API：

- MATLAB 数据 API 支持 MATLAB 数据类型和优化，例如传递给 MEX 函数的数据数组的写入时复制。有关详细信息，请参阅“MATLAB Data API for C++”。
- MATLAB C++ 引擎 API 的子集支持调用 MATLAB 函数、在 MATLAB 工作区中执行语句，以及访问变量和对象。有关详细信息，请参阅“C++ MEX API” （第 9-10 页）。

C++ MEX API 支持 C++11 功能，并且与 C MEX API 不兼容。您不能在 MEX 文件中混用这些 API。

C++ MEX 函数的基本设计

C++ MEX 函数作为从 **matlab::mex::Function** 继承的名为 **MexFunction** 的类来实现。**MexFunction** 类将覆盖函数调用运算符 **operator()**。此实现会创建一个可以像函数一样调用的函数对象。

从 MATLAB 中调用 MEX 函数将实例化此函数对象，其状态在对同一个 MEX 函数的后续调用中保持不变。

下面是 C++ MEX 函数的基本设计。它是 **matlab::mex::Function** 的子类，必须命名为 **MexFunction**。**MexFunction** 类将覆盖函数调用运算符 **operator()**。

```
#include "mex.hpp"
#include "mexAdapter.hpp"

class MexFunction : public matlab::mex::Function {
public:
    void operator()(matlab::mex::ArgumentList outputs, matlab::mex::ArgumentList inputs) {
        // Function implementation
        ...
    }
};
```

MEX 函数的输入和输出作为 **matlab::mex::ArgumentList** 中的元素进行传递。每个输入或输出参数都是 **matlab::mex::ArgumentList** 中包含的一个 **matlab::data::Array**。

有关示例，请参阅“创建 C++ MEX 源文件” （第 9-4 页）。

从 MATLAB 中调用 MEX 函数

要调用 MEX 函数，请使用该文件的名称，但不带文件扩展名。调用语法取决于 MEX 函数定义的输入和输出参数。调用时，MEX 文件必须位于 MATLAB 路径或当前工作文件夹中。

C++ MEX 函数示例

以下示例说明了 C++ MEX 函数的实现：

- **arrayProduct.cpp** - 将数组乘以标量输入并返回生成的数组。
- **yprime.cpp** - 定义受限三体问题的微分方程。

- **phonebook.cpp** - 说明如何操作结构体。
- **modifyObjectProperty.cpp** - 说明如何与 MATLAB 对象一起使用。

另请参阅

相关示例

- “编译 C++ MEX 程序” (第 9-8 页)
- “C++ MEX API” (第 9-10 页)
- “Structure of C++ MEX Function”
- “编写可从 MATLAB（MEX 文件）调用的 C++ 函数”

创建 C++ MEX 源文件

下面说明如何创建基本的 C++ MEX 函数。此函数只向输入数组的每个元素添加偏移量，以演示基本的输入和输出。有关创建 MEX 函数源代码的更详细讨论，请参阅 “Structure of C++ MEX Function” 和相关主题。

创建源文件

使用您的编辑器，创建扩展名为 **.cpp** 的文件，并添加注释说明。例如，**MyMEXFunction.cpp**。

```
/* MyMEXFunction
 * c = MyMEXFunction(a,b);
 * Adds offset argument a to each element of double array b and
 * returns the modified array c.
*/
```

添加所需的头文件

对于 C++ MEX 函数，添加下列头文件。

```
/* MyMEXFunction
 * c = MyMEXFunction(a,b);
 * Adds offset argument a to each element of double array b and
 * returns the modified array c.
*/

#include "mex.hpp"
#include "mexAdapter.hpp"
```

使用便利定义

（可选）为 **matlab::data** 指定命名空间并定义其他便利。

```
/* MyMEXFunction
 * c = MyMEXFunction(a,b);
 * Adds offset argument a to each element of double array b and
 * returns the modified array c.
*/

#include "mex.hpp"
#include "mexAdapter.hpp"

using namespace matlab::data;
using matlab::mex::ArgumentList;
```

定义 MexFunction 类

所有 C++ MEX 函数都作为名为 **MexFunction** 的类来实现。此类必须派生自 **matlab::mex::Function**。

```
/* MyMEXFunction
 * c = MyMEXFunction(a,b);
 * Adds offset argument a to each element of double array b and
 * returns the modified array c.
```

```
*/

#include "mex.hpp"
#include "mexAdapter.hpp"

using namespace matlab::data;
using matlab::mex::ArgumentList;

class MexFunction : public matlab::mex::Function {

};
```

定义 operator()

所有 **MexFunction** 类必须覆盖函数调用运算符 **operator()**，以接受 **matlab::mex::ArgumentList** 类型的两个参数。这些参数包含输入和输出。

```
/* MyMEXFunction
 * c = MyMEXFunction(a,b);
 * Adds offset argument a to each element of double array b and
 * returns the modified array c.
*/

#include "mex.hpp"
#include "mexAdapter.hpp"

using namespace matlab::data;
using matlab::mex::ArgumentList;

class MexFunction : public matlab::mex::Function {
public:
   void operator()(ArgumentList outputs, ArgumentList inputs) {

   }

};
```

添加成员函数以检查参数

进行测试，以查看参数的类型和大小是否正确。如果测试失败，请调用 MATLAB **error** 函数。

```
/* MyMEXFunction
 * c = MyMEXFunction(a,b);
 * Adds offset argument a to each element of double array b and
 * returns the modified array c.
*/

#include "mex.hpp"
#include "mexAdapter.hpp"

using namespace matlab::data;
using matlab::mex::ArgumentList;

class MexFunction : public matlab::mex::Function {
public:
   void operator()(ArgumentList outputs, ArgumentList inputs) {

   }
   void checkArguments(ArgumentList outputs, ArgumentList inputs) {
      // Get pointer to engine
      std::shared_ptr<matlab::engine::MATLABEngine> matlabPtr = getEngine();

      // Get array factory
```

```
        ArrayFactory factory;

        // Check offset argument: First input must be scalar double
        if (inputs[0].getType() != ArrayType::DOUBLE ||
            inputs[0].getNumberOfElements() != 1)
        {
            matlabPtr->feval(u"error",
                0,
                std::vector<Array>({ factory.createScalar("First input must be scalar double") }));
        }

        // Check array argument: Second input must be double array
        if (inputs[1].getType() != ArrayType::DOUBLE)
        {
            matlabPtr->feval(u"error",
                0,
                std::vector<Array>({ factory.createScalar("Input must be double array") }));
        }
        // Check number of outputs
        if (outputs.size() > 1)
        {
            matlabPtr->feval(u"error",
                0,
                std::vector<Array>({ factory.createScalar("Only one output is returned") }));
        }
    }
};
```

实现计算

获取标量偏移量，并将其赋给 **const double**。获取输入数组，并将其移至 **matlab::data::TypedArray<double>** 来处理该数组。将偏移量添加到数组中的每个元素，并将修改后的数组赋给输出变量。

```
/* MyMEXFunction
 * c = MyMEXFunction(a,b);
 * Adds offset argument a to each element of double array b and
 * returns the modified array c.
*/

#include "mex.hpp"
#include "mexAdapter.hpp"

using namespace matlab::data;
using matlab::mex::ArgumentList;

class MexFunction : public matlab::mex::Function {
public:
    void operator()(ArgumentList outputs, ArgumentList inputs) {
        checkArguments(outputs, inputs);
        const double offSet = inputs[0][0];
        TypedArray<double> doubleArray = std::move(inputs[1]);
        for (auto& elem : doubleArray) {
            elem += offSet;
        }
        outputs[0] = doubleArray;
    }

    void checkArguments(ArgumentList outputs, ArgumentList inputs) {
        // Get pointer to engine
        std::shared_ptr<matlab::engine::MATLABEngine> matlabPtr = getEngine();

        // Get array factory
        ArrayFactory factory;

        // Check offset argument: First input must be scalar double
        if (inputs[0].getType() != ArrayType::DOUBLE ||
            inputs[0].getType() == ArrayType::COMPLEX_DOUBLE ||
            inputs[0].getNumberOfElements() != 1)
        {
            matlabPtr->feval(u"error",
                0,
                std::vector<Array>({ factory.createScalar("First input must be scalar double") }));
        }

        // Check array argument: Second input must be double array
        if (inputs[1].getType() != ArrayType::DOUBLE ||
            inputs[1].getType() == ArrayType::COMPLEX_DOUBLE)
```

```
        {
            matlabPtr->feval(u"error",
                0,
                std::vector<Array>({ factory.createScalar("Input must be double array") }));
        }
        // Check number of outputs
        if (outputs.size() > 1) {
            matlabPtr->feval(u"error",
                0,
                std::vector<Array>({ factory.createScalar("Only one output is returned") }));
        }
    }
};
```

设置和编译

安装受支持的编译器后，使用 **mex** 命令编译您的 MEX 函数。

```
mex -setup c++
mex MyMEXFunction.cpp
```

有关编译 MEX 函数的详细信息，请参阅“编译 C++ MEX 程序” （第 9-8 页）。

调用 MEX 函数

从 MATLAB 调用您的 MEX 函数。

```
b = MyMEXFunction(11.5, rand(1000));
```

另请参阅

mex | **matlab::mex::Function** | **matlab::mex::ArgumentList**

相关示例

- “编译 C++ MEX 程序” （第 9-8 页）
- “C++ MEX API” （第 9-10 页）
- “编写可从 MATLAB（MEX 文件）调用的 C++ 函数”

编译 C++ MEX 程序

使用 MATLAB **mex** 编译您的 C++ MEX 应用程序，以设置您的环境并编译 C++ 源代码。

支持的编译器

使用支持 C++11 的编译器。有关支持的编译器的最新列表，请参阅支持和兼容的编译器网站。

使用 mex 命令编译 .cpp 文件

如果您安装了受支持的编译器之一，请使用 **mex** 命令为 C++ MEX 应用程序设置编译器。当提供了用于选择编译器的选项时，选择 MATLAB C++ MEX 支持的一个已安装的编译器。

```
mex -setup C++
```

使用 MATLAB **mex** 命令编译您的 C++ MEX 程序。

```
mex MyMEXCode.cpp
```

MEX 包含文件

在您的 C++ 源代码中包含以下头文件。头文件包含函数声明以及您在 API 库中访问的例程的原型。这些文件位于 **matlabroot/extern/include** 文件夹中，并且与用于 Windows、Mac 和 Linux 系统的文件相同。C++ MEX 文件使用以下头文件：

- **mex.hpp** - C++ MEX API 的定义
- **mexAdapter.hpp** - C++ MEX 函数运算符所需的实用工具

注意 在跨多个文件的 MEX 应用程序中，将 **mexAdapter.hpp** 与 **MexFunction** 类定义仅包含一次。

文件扩展名

MEX 文件是特定于平台的。MATLAB 通过特定于平台的扩展名来识别 MEX 文件。下表列出了 MEX 文件的特定于平台的扩展名。

MEX 文件与平台相关的扩展名

平台	二进制 MEX 文件扩展名
Linux（64 位）	**mexa64**
macOS（64 位）	**mexmaci64**
Windows（64 位）	**mexw64**

另请参阅

mex | **matlab::engine::MATLABEngine**

相关示例

- “C++ Engine API”

- “Test Your C++ Build Environment”
- “编写可从 MATLAB（MEX 文件）调用的 C++ 函数”

C++ MEX API

注意 C++ MEX API 与 “编写可从 MATLAB（MEX 文件）调用的 C 函数”中所述的 C MEX API 不兼容。您不能在 MEX 文件中混用这些 API。

通过 C++ MEX API，您可以创建利用 C++11 功能（例如移动语义、异常处理和内存管理）的应用程序。

- **matlab::mex::Function** - C++ MEX 函数的基类。
- **matlab::mex::ArgumentList** - C++ MEX 函数的输入和输出的容器。
- **matlab::engine::MATLABEngine** - 定义引擎 API 的类。

matlab::mex::Function 类

所有 MEX 文件实现均是从 **matlab::mex::Function** 派生的类。

matlab::mex::Function 函数	**描述**
getEngine	获取指向 **MATLABEngine** 对象的指针。
mexLock	防止从内存中清除 MEX 文件。
mexUnlock	允许从内存中清除 MEX 文件。
getFunctionName	获取当前 MEX 函数的名称。

matlab::mex::ArgumentList 类

通过 **mex::Function** 类的 **operator()** 传递的 MEX 函数参数是 **matlab::mex::ArgumentList** 容器。**ArgumentList** 完全涵盖数组的基本集合。

matlab::mex::ArgumentList 方法	**描述**
operator[ ]	支持对 **ArgumentList** 的元素进行 **[]** 索引。
begin	启动迭代器。
end	结束迭代器。
size	返回参数列表中的元素数。使用此方法检查在调用点指定的输入和输出的数目。
empty	返回逻辑值，指示参数列表是否为空 (**size() == 0**)。

C++ Engine API

使用 **matlab::engine::MATLABEngine** 类访问 MATLAB 函数、变量和对象。要调用此类中的方法，请使用 **getEngine** 创建一个共享指针，如此示例中的 **matlabPtr**：

```
std::shared_ptr<matlab::engine::MATLABEngine> matlabPtr = getEngine();
```

使用 **matlabPtr** 调用引擎方法。例如：

```
matlabPtr->feval(...);
```

仅可在 **mex::Function** 类所在相同线程上调用引擎方法。

matlab::engine::MATLABEngine 方法	**描述**	**示例**
feval	使用输入参数同步计算 MATLAB 函数。使用 **feval** 将参数从 C++ 传递给 MATLAB，并将结果从 MATLAB 返回给 C++。	“Call MATLAB Functions from MEX Functions”
fevalAsync	使用输入参数和返回值异步计算 MATLAB 函数。	有关详细信息，请参阅“Making async Requests Using mexCallMATLAB”。
eval	以同步方式计算 MATLAB 语句字符串。	“Execute MATLAB Statements from MEX Function”
evalAsync	以异步方式计算 MATLAB 语句字符串。	有关详细信息，请参阅“Making async Requests Using mexCallMATLAB”。
getVariable	从 MATLAB 基础工作区或全局工作区获取变量。	“Set and Get MATLAB Variables from MEX”
getVariableAsync	以异步方式从 MATLAB 基础工作区或全局工作区获取变量。	
setVariable	将变量放入 MATLAB 基础工作区或全局工作区中。如果 MATLAB 工作区中存在同名变量，**setVariable** 会覆盖它。	“Set and Get MATLAB Variables from MEX”
setVariableAsync	以异步方式将变量放入 MATLAB 基础工作区或全局工作区中。	
getProperty	获取对象属性的值。	“MATLAB Objects in MEX Functions”
getPropertyAsync	以异步方式获取对象属性的值。	
setProperty	设置对象属性的值。	“MATLAB Objects in MEX Functions”
setPropertyAsync	以异步方式设置对象属性的值。	

有关异常的信息，请参阅“MATLAB Engine API for C++ Exception Classes”。有关示例，请参阅“Catch Exceptions in MEX Function”。

另请参阅

相关示例

- “编写可从 MATLAB（MEX 文件）调用的 C++ 函数”

10

Fortran MEX 文件

编译 Fortran MEX 文件

此示例说明如何编译示例 MEX 文件 **timestwo**。使用此示例验证系统的编译配置。

要编译代码示例，请首先将文件复制到一个可写文件夹中，例如路径上的 **c:\work**：

```
copyfile(fullfile(matlabroot,'extern','examples','refbook','timestwo.F'),'.','f')
```

使用 **mex** 命令编译 MEX 文件。

```
mex timestwo.F
```

此命令将创建文件 **timestwo.ext**，其中 **ext** 是由 **mexext** 函数返回的值。

函数 **timestwo** 接受标量输入并使其倍增。像调用 MATLAB 函数一样调用 **timestwo**。

```
timestwo(4)
```

```
ans =
   8
```

另请参阅

mex | **mexext**

详细信息

- “Handling Large mxArrays in C MEX Files”
- “升级 MEX 文件以使用 64 位 API” （第 7-17 页）

11

通过 C/C++ 和 Fortran 程序调用 MATLAB Engine

适用于 C 和 Fortran 的 MATLAB 引擎 API

MATLAB C 和 Fortran 引擎库包含的例程允许您使用 MATLAB 作为计算引擎，从您自己的程序中调用 MATLAB。使用 MATLAB 引擎需要已安装版本的 MATLAB；您无法在只有 MATLAB Runtime 的计算机上运行 MATLAB 引擎。

引擎程序为独立程序。这些程序通过 UNIX 系统中的管道和 Microsoft Windows 系统中的 Microsoft 组件对象模型 (COM) 接口与单独的 MATLAB 进程通信。MATLAB 提供了函数库，允许您启动和结束 MATLAB 进程，在 MATLAB 中发送和接收数据，以及发送将在 MATLAB 中处理的命令。

您可以使用 MATLAB 引擎执行的一些操作包括：

- 调用数学例程，例如，通过您自己的程序反转数组或计算 FFT。当采用这种方法时，MATLAB 是一个功能强大且可编程的数学子例程库。
- 针对特定任务编译整个系统。例如，前端（用户界面）采用 C 编程，后端（分析）采用 MATLAB 编程。

MATLAB 引擎的工作方式是通过您自己的程序以单独进程的方式在后台运行。一些优势包括：

- 在 UNIX 系统中，该引擎可以在您的计算机或您网络中的其他任何 UNIX 计算机上运行，包括不同架构的计算机。利用此配置，您可以在工作站上实现用户界面，并在位于网络内任何其他位置的速度更快的计算机上执行计算。有关详细信息，请参阅 **engOpen** 参考页。
- 该引擎不要求您的程序链接到完整的 MATLAB 程序（大量的代码），而是链接到一个较小的引擎库。

MATLAB 引擎无法读取采用基于 HDF5 的格式的 MAT 文件。这类 MAT 文件使用 **save** 函数的 **-v7.3** 选项保存数据，或使用 C 或 Fortran **matOpen** 函数的 **w7.3** 模式参数打开。

注意 要在 UNIX 平台上运行 MATLAB 引擎，您必须在 **/bin/csh** 安装 C shell **csh**。

与 MATLAB 软件通信

在 UNIX 系统中，引擎库使用管道与引擎通信，并在需要时使用 **rsh** 来远程执行。在 Microsoft Windows 系统中，引擎库使用组件对象模型 (COM) 接口与引擎通信。

另请参阅

相关示例

- “从 C 应用程序中调用 MATLAB 函数” （第 11-3 页）
- “从 Fortran 应用程序中调用 MATLAB 函数” （第 11-5 页）

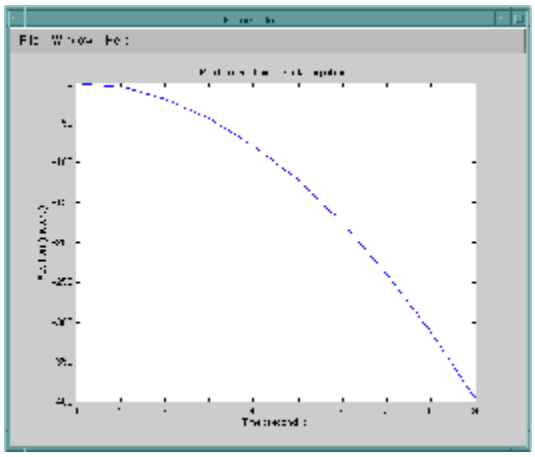
从 C 应用程序中调用 MATLAB 函数

matlabroot/extern/examples/eng_mat 文件夹中的程序 **engdemo.c** 说明如何通过独立的 C 程序调用引擎函数。此示例使用 “C Matrix API”。

注意 要从 C++ 应用程序调用 MATLAB 函数，请使用 “MATLAB Data API for C++”。有关详细信息，请参阅“从 C++ 调用 MATLAB 函数” （第 14-4 页）。

对于该程序的 Microsoft Windows 版本，请打开 **matlabroot\extern\examples\eng_mat** 文件夹中的 **engwindemo.c**。对于 C++ 版本，请打开 **engdemo.cpp**。

该程序的第一部分将启动 MATLAB 并向其发送数据。MATLAB 将分析数据并绘制结果图。



程序继续运行：

```
Press Return to continue
```

按 **Return** 继续执行程序：

```
Done for Part I.
Enter a MATLAB command to evaluate.  This command should
create a variable X.  This program will then determine
what kind of variable you created.
For example: X = 1:5
```

输入 **X = 17.5** 继续执行程序。

```
X = 17.5

X =

   17.5000


Retrieving X...
X is class double
Done!
```

最后，程序释放内存，关闭 MATLAB 引擎，然后退出。

另请参阅

相关示例

- “Build Windows Engine Application”
- “Build Engine Application on Linux”

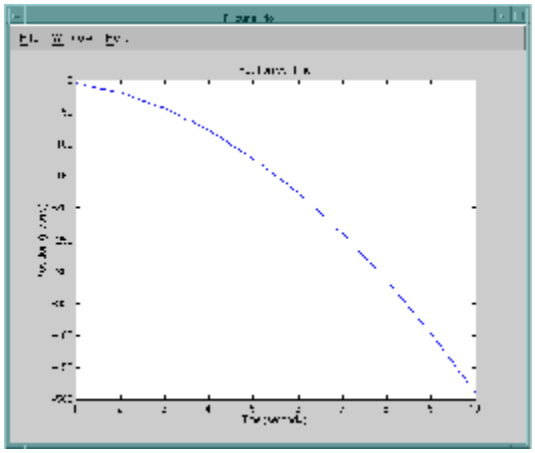
详细信息

- “使用 IDE 编译引擎应用程序” (第 11-8 页)

从 Fortran 应用程序中调用 MATLAB 函数

matlabroot/extern/examples/eng_mat 文件夹中的程序 **fengdemo.F** 说明如何通过独立的 Fortran 程序调用引擎函数。要查看代码，请打开此文件。

执行此程序会启动 MATLAB，向其发送数据，并绘制结果。



程序继续运行：

```
Type 0 <return> to Exit
Type 1 <return> to continue
```

在提示符下输入 1 会继续程序执行：

```
1
 MATLAB computed the following distances:
   time(s)  distance(m)
   1.00     -4.90
   2.00     -19.6
   3.00     -44.1
   4.00     -78.4
   5.00     -123.
   6.00     -176.
   7.00     -240.
   8.00     -314.
   9.00     -397.
   10.0     -490.
```

最后，程序释放内存，关闭 MATLAB 引擎，然后退出。

另请参阅

详细信息

- “Build and Run Fortran Engine Applications on Windows”
- “Build and Run Fortran Engine Applications on Linux”
- “Build and Run Fortran Engine Applications on macOS”

在 Linux 系统上设置运行时库路径

在运行时，通过设置环境变量 **LD_LIBRARY_PATH**，告诉操作系统 API 共享库所在的位置。将值设置为 **matlabroot/bin/glnxa64:matlabroot/sys/os/glnxa64**。

使用的命令取决于您的 shell。该命令将替换现有的 **LD_LIBRARY_PATH** 值。如果已定义 **LD_LIBRARY_PATH**，则将新值添加到现有值之前。

如果您的系统上安装了多个版本的 MATLAB，则用于编译引擎应用程序的版本必须是系统 **Path** 环境变量中列出的第一个版本。否则，MATLAB 将显示 **Can't start MATLAB engine**。

可以在每次运行 MATLAB 时设置路径，也可以将命令放在 MATLAB 启动脚本中。

C shell

按照以下命令格式设置库路径。

```
setenv LD_LIBRARY_PATH matlabroot/bin/glnxa64:matlabroot/sys/os/glnxa64
```

您可以将这些命令放在启动脚本中，如 **~/.cshrc**。

Bourne shell

按照以下命令格式设置库路径。

```
LD_LIBRARY_PATH=matlabroot/bin/glnxa64:matlabroot/sys/os/glnxa64:LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

您可以将这些命令放在一个启动脚本中，如 **~/.profile**。

另请参阅

matlabroot

在 macOS 系统上设置运行时库路径

在运行时，通过设置环境变量 **DYLD_LIBRARY_PATH**，告诉操作系统 API 共享库所在的位置。将值设置为 **matlabroot/bin/maci64:matlabroot/sys/os/maci64**。

使用的命令取决于您的 shell。该命令将替换现有的 **DYLD_LIBRARY_PATH** 值。如果已定义 **DYLD_LIBRARY_PATH**，则将新值添加到现有值之前。

如果您的系统上安装了多个版本的 MATLAB，则用于编译引擎应用程序的版本必须是系统 **Path** 环境变量中列出的第一个版本。否则，MATLAB 将显示 **Can't start MATLAB engine**。

可以在每次运行 MATLAB 时设置路径，也可以将命令放在 MATLAB 启动脚本中。

C shell

按照以下命令格式设置库路径。

```
setenv DYLD_LIBRARY_PATH matlabroot/bin/maci64:matlabroot/sys/os/maci64
```

例如，对于 Mac 系统上的 MATLAB R2015a：

```
setenv DYLD_LIBRARY_PATH /Applications/MATLAB_R2015a.app/bin/maci64:/Applications/MATLAB_R2015a.app/sys/os/maci64
```

您可以将这些命令放在启动脚本中，如 **~/.cshrc**。

Bourne shell

按照以下命令格式设置库路径。

```
DYLD_LIBRARY_PATH=matlabroot/bin/maci64:matlabroot/sys/os/maci64:$DYLD_LIBRARY_PATH
export DYLD_LIBRARY_PATH
```

例如，对于 Mac 系统上的 MATLAB R2015a：

```
DYLD_LIBRARY_PATH=/Applications/MATLAB_R2015a.app/bin/maci64:/Applications/MATLAB_R2015a.app/sys/os/maci64:$DYLD_LIBRARY_PATH
export DYLD_LIBRARY_PATH
```

您可以将这些命令放在一个启动脚本中，如 **~/.profile**。

另请参阅

matlabroot

外部网站

- 向 MAC 中的 DYLD_LIBRARY_PATH 追加库路径

使用 IDE 编译引擎应用程序

您可以使用 MATLAB 编辑器编写您的引擎应用代码，并使用 **mex** 命令编译它。如果您更喜欢使用集成开发环境 (IDE)（如 Microsoft Visual Studio 或 Xcode）来编写源代码，您仍可以使用 **mex** 命令。但是，要使用 IDE 编译您的应用程序，请遵循以下主题中的指导原则。

配置 IDE

要使用集成开发环境编译引擎应用程序，您的 IDE 需要 MATLAB 支持的编译器。有关支持的编译器的最新列表，请参阅支持和兼容的编译器。

引擎应用程序需要引擎库 **libeng**、Matrix Library **libmx** 和支持 include 文件。当您使用 **mex** 命令编译时，MATLAB 被配置为查找这些文件。在您的 IDE 中编译时，您必须配置 IDE 以查找它们。这些设置的具体位置取决于您的 IDE。请参阅您的产品文档。

引擎 Include 文件

头部文件包含函数声明以及您在 API 库中访问的例程的原型。这些文件位于 **matlabroot\extern\include** 文件夹中，并且与用于 Windows、macOS 和 Linux 系统的文件相同。引擎应用程序使用：

- **engine.h** - 引擎例程的函数原型。
- **matrix.h** - 矩阵访问例程的 **mxArray** 结构体和函数原型的定义。
- **mat.h**（可选）- **mat** 例程的函数原型

在 IDE 中，将预处理器包含路径设置为由以下 MATLAB 命令返回的值：

```
fullfile(matlabroot,'extern','include')
```

引擎库

您需要 **libeng** 和 **libmx** 共享库。文件的名称是特定于平台的。将这些库名称添加到您的 IDE 配置中。有关说明，请参阅您的 IDE 产品文档。

Windows 库

在这些路径设定中，将 **compiler** 替换为 **microsoft** 或 **mingw64**。

- 引擎库 - **matlabroot\extern\lib\win64\compiler\libeng.lib**
- 矩阵库 - **matlabroot\extern\lib\win64\compiler\libmx.lib**
- MEX 库（可选）- **matlabroot\extern\lib\win64\compiler\libmex.lib**
- MAT 文件库（可选）- **matlabroot\extern\lib\win64\compiler\libmat.lib**

Linux 库

- 引擎库 - **matlabroot/bin/glnxa64/libeng.so**
- 矩阵库 - **matlabroot/bin/glnxa64/libmx.so**
- MEX 库（可选）- **matlabroot/bin/glnxa64/libmex.so**
- MAT 文件库（可选）- **matlabroot/bin/glnxa64/libmat.so**

macOS 库

- 引擎库 - **matlabroot/bin/maci64/libeng.dylib**
- 矩阵库 - **matlabroot/bin/maci64/libmx.dylib**
- MEX 库（可选）- **matlabroot/bin/maci64/libmex.dylib**
- MAT 文件库（可选）- **matlabroot/bin/maci64/libmat.dylib**

另请参阅

相关示例

- “Build Windows Engine Application”
- “Build Engine Application on Linux”
- How can I compile a MATLAB Engine application using Microsoft Visual Studio 9.0 or 10.0?
- How can I build an Engine application using the Xcode IDE on Mac?

Engine API for Java

用于 Java 的 MATLAB 引擎 API

用于 Java 的 MATLAB 引擎 API 使 Java 程序能够与 MATLAB 进行同步或异步交互，包括：

- 启动和终止 MATLAB。
- 连接到本地计算机上的 MATLAB 会话和与之断开连接。
- 使用从 Java 传递的输入参数和从 MATLAB 返回的输出变量调用 MATLAB。
- 对 MATLAB 基础工作区中的 MATLAB 语句进行求值。
- 将变量从 Java 传递给 MATLAB 和从 MATLAB 传递给 Java。

与 MATLAB 的异步通信基于 Java Future 接口 java.util.concurrent.Future。

在 Java 和 MATLAB 之间传递的数据数组的大小限制为不超过 2 GB。此限制应用于进程间传递的数据和支持信息。

用于 Java 的 MATLAB 引擎 API 作为 MATLAB 产品的一部分提供。您必须安装支持的 JDK™ 版本才能编译用于 Java 的 MATLAB 引擎应用程序。有关版本信息，请参阅其他语言的 MATLAB 接口。

另请参阅

相关示例

- “Java Engine API Summary”
- “编译 Java 引擎程序” （第 12-3 页）
- “Java Example Source Code”
- “Start and Close MATLAB Session from Java”

编译 Java 引擎程序

本节内容

一般要求

要为编译引擎应用程序而设置您的 Java 环境，需要满足以下要求：

- 将 **matlabroot/extern/engines/java/jar/engine.jar** 添加到您的 Java 类路径中。
- 使用支持的 JDK 版本编译引擎应用程序。有关版本信息，请参阅其他语言的 MATLAB 接口。
- 确保您的 JRE™ 版本不早于您的 JDK 版本。

要运行 Java，请将 **matlabroot/bin/<arch>** 文件夹添加到您的系统环境变量中。**<arch>** 是您的计算机架构。例如，**win64** 适用于 64 位 Microsoft Windows 机器，macOS 上的 **maci64** 或 Linux 上的 **glnxa64**。

matlabroot 是由 MATLAB **matlabroot** 命令返回的值。该命令返回安装 MATLAB 的文件夹。

下表列出了环境变量的名称和路径的值。

操作系统	变量	路径
Windows	**PATH**	**matlabroot\extern\bin\win64**
macOS	**DYLD_LIBRARY_PATH**	**matlabroot/extern/bin/maci64**
Linux	**LD_LIBRARY_PATH**	**matlabroot/extern/bin/glnxa64:matlabroot/sys/os/glnxa64**

在 Windows 上编译并运行 Java 代码

编译您的 Java 代码：

```
javac -classpath matlabroot\extern\engines\java\jar\engine.jar MyJavaCode.java
```

运行 Java 程序：

```
java -classpath .;matlabroot\extern\engines\java\jar\engine.jar MyJavaCode
```

设置系统路径

要从 Windows 命令提示符设置运行时库路径，请键入以下命令。

```
set PATH=matlabroot\bin\win64;%PATH%
```

每次您打开 Windows 命令处理器时都要设置该路径。

您还可以从“系统属性”对话框中设置 **PATH** 变量。从**控制面板** > **系统** > **高级系统设置** > **高级**选项卡上，点击**环境变量**。在**系统变量**下，选择 **Path** 并点击**编辑**。通过插入 **matlabroot\bin\win64** 修改 **Path**；在**变量 值**的开头位置。点击**确定**关闭对话框，然后关闭**控制面板**对话框。

在 macOS 上编译并运行 Java 代码

用于 Java 的 MATLAB 引擎 API 仅支持 macOS 系统上的 **maci64**。

编译 Java 代码：

```
javac -classpath matlabroot/extern/engines/java/jar/engine.jar MyJavaCode.java
```

指定 Java 库路径和运行程序

在一条语句中指定 Java 库路径并运行 Java 程序。

```
java -Djava.library.path=matlabroot/bin/maci64 -classpath .:matlabroot/extern/engines/java/jar/engine.jar MyJavaCode
```

设置系统变量并运行程序

设置 **DYLD_LIBRARY_PATH** 变量并运行 Java 程序。例如，使用 C shell：

```
setenv DYLD_LIBRARY_PATH matlabroot/bin/maci64:$DYLD_LIBRARY_PATH
java -classpath .:matlabroot/extern/engines/java/jar/engine.jar MyJavaCode
```

从 C shell 设置变量

您可以将这些命令放在一个启动脚本中，如 **~/.cshrc**。

```
setenv DYLD_LIBRARY_PATH matlabroot/bin/maci64:$DYLD_LIBRARY_PATH
```

在 Bourne shell 中设置变量

您可以将这些命令放在一个启动脚本中，如 **~/.profile**。

```
DYLD_LIBRARY_PATH=matlabroot/bin/maci64:$DYLD_LIBRARY_PATH
export DYLD_LIBRARY_PATH
```

使用 Java 1.8.0 版的早期内部版本

在使用 Java 1.8.0 版的早期内部版本（例如 1.8.0_111）时，可能无法识别 DYLD_LIBRARY_PATH 环境变量。如果出现 **java.lang.UnsatisfiedLinkError** 异常，请显式设置 **java.library.path**：

```
java -Djava.library.path=matlabroot/bin/maci64 -classpath .:matlabroot/extern/engines/java/jar/engine.jar MyJavaCode
```

在 Linux 上编译并运行 Java 代码

用于 MATLAB 的 Java 引擎 API 仅支持 Linux 系统上的 **glnxa64**。

编译 Java 代码：

```
javac -classpath matlabroot/extern/engines/java/jar/engine.jar MyJavaCode.java
```

指定 Java 库路径和运行程序

如果兼容的 GCC 库位于搜索路径中，则可以在 Java 库搜索路径中添加 **matlabroot/bin/glnxa64** 并运行示例，而不必设置 **LD_LIBRARY_PATH** 变量。有关支持的编译器的信息，请参阅支持和兼容的编译器。

在一条语句中指定 Java 库路径并运行 Java 程序。

```
java -Djava.library.path=matlabroot/bin/glnxa64 -classpath .:matlabroot/extern/engines/java/jar/engine.jar MyJavaCode
```

设置系统变量并运行程序

设置 **LD_LIBRARY_PATH** 变量并运行 Java 程序。例如，使用 C shell：

```
setenv LD_LIBRARY_PATH matlabroot/bin/glnxa64:matlabroot/sys/os/glnxa64:$LD_LIBRARY_PATH
java -classpath .:matlabroot/extern/engines/java/jar/engine.jar MyJavaCode
```

从 C shell 设置变量

您可以将这些命令放在一个启动脚本中，如 **~/.cshrc**。

```
setenv LD_LIBRARY_PATH matlabroot/bin/glnxa64:matlabroot/sys/os/glnxa64:$LD_LIBRARY_PATH
```

从 Bourne shell 设置变量

您可以将这些命令放在一个启动脚本中，如 **~/.profile**。

```
LD_LIBRARY_PATH=matlabroot/bin/glnxa64:matlabroot/sys/os/glnxa64:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

另请参阅

相关示例

- “Java Engine API Summary”

从 Java 执行 MATLAB 函数

本节内容

调用 MATLAB 函数

您可以使用 **MatlabEngine feval** 和 **fevalAsync** 方法从 Java 执行 MATLAB 函数。这些方法的工作方式类似于 MATLAB **feval** 函数。如果要将函数执行的结果返回给 Java 或从 Java 传递参数，请使用 **feval** 和 **fevalAsync**。

要调用 MATLAB 函数，需要满足以下条件：

- 将函数名称作为字符串传递。
- 定义 MATLAB 函数所需的输入参数。
- 指定 MATLAB 函数所需的输出的数量（如果未指定，则假定为 1）。
- 为 MATLAB 函数的输出定义适当的返回类型。
- 使用写入器将输出从 MATLAB 命令行窗口重定向到 Java。

您还可以使用 **MatlabEngine eval** 和 **evalAsync** 方法来计算 MATLAB 表达式。这些方法使您能够在 MATLAB 工作区中创建变量，但不返回值。

使用单一返回参数执行函数

以下示例代码使用 MATLAB **sqrt** 函数求双精度数组中元素的平方根。**feval** 方法返回包含 **sqrt** 函数调用结果的双精度数组。

```
import com.mathworks.engine.*;

public class javaFevalFunc{
   public static void main(String[] args) throws Exception{
      MatlabEngine eng = MatlabEngine.startMatlab();
      double[] a = {2.0 ,4.0, 6.0};
      double[] roots = eng.feval("sqrt", a);
      for (double e: roots) {
         System.out.println(e);
      }
      eng.close();
   }
}
```

使用多个返回参数执行函数

以下示例代码使用 MATLAB **gcd** 函数对作为输入参数传递的两个整数值求最大公约数和 Bézout 系数。**feval** 方法返回包含 **gcd** 函数调用结果的对象数组。返回值是整数。

由于 MATLAB **gcd** 函数返回三个输出参数，请将返回值的数量指定为 **feval** 方法的第一个参数。

```
import com.mathworks.engine.*;

public class javaFevalFcnMulti {
   public static void main(String[] args) throws Exception {
      MatlabEngine eng = MatlabEngine.startMatlab();
      Object[] results = eng.feval(3, "gcd", 40, 60);
      Integer G = (Integer)results[0];
      Integer U = (Integer)results[1];
      Integer V = (Integer)results[2];
      eng.close();
   }
}
```

何时指定输出参数的数量

使用 **MatlabEngine feval** 和 **fevalAsync** 方法可以指定由 MATLAB 函数返回的输出参数的数量。默认情况下，来自 MATLAB 函数的输出参数的数量假定为 **1**。

如果要调用 MATLAB 函数且不提供输出或提供多个输出，请将该输出数量指定为传递给 **feval** 或 **fevalAsync** 的第一个参数。

例如，以下代码使用带有三个输出参数的语法调用 MATLAB **gcd** 函数：

```
Object[] results = eng.feval(3, "gcd", 40, 60);
```

根据请求的输出数量，MATLAB 函数的行为可能会有所不同。某些函数不返回任何输出或返回指定数量的输出。例如，MATLAB **pause** 函数使执行暂停指定的秒数。但是，如果您使用一个输出参数调用 **pause**，此函数将立即返回状态值。因此，以下代码不会导致 MATLAB 暂停，原因是 **feval** 请求一个输出参数。

```
eng.feval("pause", 10);
```

要使 MATLAB 执行暂停 10 秒，请将输出数量指定为 **0**。

```
eng.feval(0, "pause", 10);
```

注意 要确保调用 MATLAB 函数而不提供输出，请将返回参数的数量指定为 **0**。

另请参阅

相关示例

- “Evaluate MATLAB Statements from Java”

从 Java 运行 Simulink 仿真

本节内容

用于运行仿真的 MATLAB 命令

您可以使用适用于 Java 的 MATLAB 引擎 API 运行 Simulink 仿真。以下是以编程方式运行仿真的基本步骤。

- 创建一个 MATLAB 引擎对象，并启动一个 MATLAB 会话。
- 在 MATLAB 中加载 Simulink 模型 (**load_system**)。
- 使用特定的仿真参数运行仿真 (**sim**)。
- 使用返回的 **Simulink.SimulationOuput** 对象的方法访问仿真结果。

有关以编程方式从 MATLAB 运行仿真的信息，请参阅“运行单个仿真” (Simulink)。

从 Java 运行 vdp 模型

Simulink **vdp** 模块图对 van der Pol 方程进行仿真。该方程为二阶微分方程。该模型使用由模型定义的初始条件和配置参数来求解方程。

运行仿真的 MATLAB 代码

以下 MATLAB 代码显示以编程方式运行仿真的命令。**Simulink.SimulationOuput** 对象的 **get** 方法返回结果和时间向量。

```
mdl = 'vdp';
load_system(mdl);
simOut = sim(mdl,'SaveOutput','on',...
  'OutputSaveName','yOut',...
  'SaveTime','on',...
  'TimeSaveName','tOut');
y = simOut.get('yOut');
t = simOut.get('tOut');
```

绘制数据图

以下 MATLAB 代码创建仿真输出的图，并将此图导出为 JPEG 图像文件。

```
plot(t,y)
print('vdpPlot','-djpeg')
```

运行仿真的 Java 代码

以下 Java 代码运行 Simulink **vdp** 模型仿真，并将结果返回到 Java。该实现执行以下操作：

- 创建一个 MATLAB 引擎对象，并启动一个 MATLAB 会话。
- 调用 MATLAB **load_system** 命令以启动 Simulink 并以异步方式加载 **vdp** 模型。轮询任务，直到 **Future** 返回。

- 调用 MATLAB **sim** 命令以设置仿真参数并运行仿真。轮询任务，直到 **Future** 返回。
- 捕获仿真结果。**sim** 函数的输出是在 MATLAB 基础工作区中创建的 MATLAB **Simulink.SimulationOuput** 对象。

  引擎 API 不支持这种类型的对象。因此，此示例使用对象的 **get** 方法访问 MATLAB 工作区中的仿真数据。
- 创建仿真数据的图，并将此图导出为 JPEG 文件。
- 将仿真结果和时间向量作为 **double** 数组返回到 Java。

```
import com.mathworks.engine.*;
import java.util.concurrent.Future;
import java.util.Arrays;

public class RunSimulation {
   public static void main(String[] args) throws Exception {
      MatlabEngine eng = MatlabEngine.startMatlab();
      Future<Void>  fLoad = eng.evalAsync("load_system('vdp')");
      while (!fLoad.isDone()){
         System.out.println("Loading Simulink model...");
         Thread.sleep(10000);
      }
      Future<Void> fSim = eng.evalAsync("simOut = sim('vdp','SaveOutput'," +
            "'on','OutputSaveName','yOut'," +
            "'SaveTime','on','TimeSaveName','tOut');");
      while (!fSim.isDone()) {
         System.out.println("Running Simulation...");
         Thread.sleep(10000);
      }
      // Get simulation data
      eng.eval("y = simOut.get('yOut');");
      eng.eval("t = simOut.get('tOut');");
      // Graph results and create image file
      eng.eval("plot(t,y)");
      eng.eval("print('vdpPlot','-djpeg')");
      // Return results to Java
      double[][] y = eng.getVariable("y");
      double[] t = eng.getVariable("t");
      // Display results
      System.out.println("Simulation result " + Arrays.deepToString(y));
      System.out.println("Time vector " + Arrays.toString(t));
      eng.close();
   }
}
```

另请参阅

相关示例

- “Evaluate MATLAB Statements from Java”
- “Pass Variables from Java to MATLAB”
- “Pass Variables from MATLAB to Java”

MATLAB Engine for Python 主题

用于 Python 的 MATLAB 引擎 API 快速入门

用于 Python® 的 MATLAB 引擎 API 提供了名为 **matlab** 的 Python 包，使您能够通过 Python 调用 MATLAB 函数。该包仅安装一次，然后您便可在当前或未来的 Python 会话中调用引擎。有关安装或启动引擎的帮助，请参阅：

- “安装用于 Python 的 MATLAB Engine API” (第 13-4 页)
- “启动和停止用于 Python 的 MATLAB 引擎” (第 13-7 页)

matlab 包中包含以下内容：

- 用于 Python 的 MATLAB 引擎 API
- Python 中的一组 MATLAB 数组类（请参阅 “MATLAB 数组作为 Python 变量” (第 13-23 页)）

引擎提供了调用 MATLAB 的函数，数组类则提供了函数来创建 Python 对象形式的 MATLAB 数组。您可以创建一个引擎并使用 **matlab.engine** 调用 MATLAB 函数。您可以在 Python 中通过调用数组类型的构造函数（例如使用 **matlab.double** 创建双精度值数组）来创建 MATLAB 数组。MATLAB 数组可以作为使用该引擎调用的 MATLAB 函数的输入参数。

下表显示了 **matlab** 包的结构。

包	函数或类	描述
matlab.engine	**start_matlab()**	Python 函数，用于创建 **MatlabEngine** 对象，并将其附加到新的 MATLAB 进程
matlab.engine	**MatlabEngine**	Python 类，用于提供调用 MATLAB 函数的方法
matlab.engine	**FutureResult**	Python 类，用于保留以异步方式调用的 MATLAB 函数的结果
matlab	**double**	Python 类，用于保留 **double** 类型的 MATLAB 数组
matlab	**single**	Python 类，用于保留 **single** 类型的 MATLAB 数组
matlab	**int8**	Python 类，用于保留 **int8** 类型的 MATLAB 数组
matlab	**int16**	Python 类，用于保留 **int16** 类型的 MATLAB 数组
matlab	**int32**	Python 类，用于保留 **int32** 类型的 MATLAB 数组

包	函数或类	描述
matlab	**int64**	Python 类，用于保留 **int64** 类型的 MATLAB 数组
matlab	**uint8**	Python 类，用于保留 **uint8** 类型的 MATLAB 数组
matlab	**uint16**	Python 类，用于保留 **uint16** 类型的 MATLAB 数组
matlab	**uint32**	Python 类，用于保留 **uint32** 类型的 MATLAB 数组
matlab	**uint64**	Python 类，用于保留 **uint64** 类型的 MATLAB 数组
matlab	**logical**	Python 类，用于保留 **logical** 类型的 MATLAB 数组
matlab	**object**	Python 类，用于保留 MATLAB 对象的句柄

另请参阅

详细信息

- “用于 Python 的 MATLAB 引擎 API 的系统要求” （第 13-35 页）

安装用于 Python 的 MATLAB Engine API

要在 Python 会话中启动 MATLAB Engine，首先必须将该引擎 API 安装为 Python 包。

验证您的配置

在安装之前，确认您的 Python 和 MATLAB 配置。

- 检查您的系统是否具有受支持的 Python 版本和 MATLAB R2014b 或更新版本。有关详细信息，请参阅 MATLAB 产品（按版本）兼容的 Python 版本。
- 要检查您的系统上是否已安装 Python，请在操作系统提示符下运行 Python。
- 将包含 Python 解释器的文件夹添加到您的路径（如果尚未在该路径中）。

安装引擎 API

您可以使用 **pip** 命令或 Python 设置脚本 **setup.py** 安装 MATLAB Engine API for Python。

使用 pip 进行安装

从 MATLAB R2022b 开始，您可以使用 **pip** 命令来安装 API。选择以下过程之一，并在系统提示符下执行。

- 要从 MATLAB 文件夹安装，请在 Windows 上键入：

  ```
  cd "matlabroot\extern\engines\python"
  python -m pip install .
  ```

- 使用以下命令从 https://pypi.org/project/matlabengine 安装引擎 API：

  ```
  python -m pip install matlabengine
  ```

使用 setup.py 进行安装

MATLAB 提供了标准的 Python **setup.py** 文件，用于使用 Python **setuptools** 编译和安装引擎。有关特定于平台的命令，请参阅“Python Setup Script to Install MATLAB Engine API”。

启动 MATLAB Engine

启动 Python。在 Python 提示符下键入以下命令，以导入 MATLAB 模块并启动引擎：

```
import matlab.engine
eng = matlab.engine.start_matlab()
```

有关详细信息，请参阅“启动和停止用于 Python 的 MATLAB 引擎” （第 13-7 页）。

用于 Python 的 MATLAB Engine API 安装故障排除

- 请确保您的 MATLAB 版本支持您的 Python 版本。请参阅 MATLAB 产品（按版本）兼容的 Python 版本。
- 确保您有管理员特权以从操作系统提示符执行安装命令。在 Windows 上，使用 **以管理员身份运行** 选项打开命令提示符。
- 您必须从指定的 MATLAB 文件夹中运行 Python 安装命令。有关详细说明，请选择 “安装引擎 API” （第 13-4 页） 中的平台链接之一。

```
python setup.py install
```

- 安装程序将引擎安装在默认的 Python 文件夹中。要使用非默认位置，请参阅“在非默认位置安装用于 Python 的 MATLAB Engine API” （第 13-6 页）。
- 如果您使用 **--prefix** 将包安装在非默认文件夹中，请确保设置 **PYTHONPATH** 环境变量。例如，假设您使用了以下安装命令：

  ```
  python setup.py install --prefix="matlab19bPy36"
  ```

  在 Python 中，使用以下命令更新 **PYTHONPATH**：

  ```
  sys.path.append("matlab19bPy36")
  ```
- 有关故障排除的详细信息，请参阅“Troubleshoot MATLAB Errors in Python”。

另请参阅

详细信息

- “用于 Python 的 MATLAB 引擎 API 的系统要求” （第 13-35 页）
- MATLAB 产品（按版本）兼容的 Python 版本
- “安装支持的 Python 实现” （第 21-10 页）
- “在非默认位置安装用于 Python 的 MATLAB Engine API” （第 13-6 页）
- “启动和停止用于 Python 的 MATLAB 引擎” （第 13-7 页）

外部网站

- Python 2.7 文档 - 安装 Python 模块

在非默认位置安装用于 Python 的 MATLAB Engine API

在非默认文件夹中编译或安装

默认情况下，安装程序将在 **matlabroot\extern\engines\python** 文件夹编译用于 Python 的引擎 API。安装程序将引擎安装在默认的 Python 文件夹中。如果您没有这些文件夹的写入权限，请选择以下非默认选项之一。如果您安装在另一个文件夹中，则创建环境变量 **PYTHONPATH**，并将值设置为该文件夹及相关子文件夹的位置。

下面是编译和安装引擎 API 的选项以及在操作系统提示符下输入的命令。

在非默认文件夹中编译，在默认文件夹中安装

如果您不具备在 MATLAB 文件夹中编译引擎的写入权限，请使用非默认文件夹 **builddir**。

```
cd "matlabroot\extern\engines\python"
python setup.py build --build-base="builddir" install
```

在默认文件夹中编译，在非默认文件夹中安装

如果您不具备在默认的 Python 文件夹中安装引擎的写入权限，请使用非默认文件夹 **installdir**。

```
cd "matlabroot\extern\engines\python"
python setup.py install --prefix="installdir"
```

要将 **installdir** 包含在 Python 包的搜索路径中，请将 **installdir** 及其相关子文件夹添加到 **PYTHONPATH** 环境变量中。

在非默认文件夹中编译和安装

如果您对 MATLAB 文件夹和默认的 Python 文件夹都没有写入权限，则可以指定非默认文件夹。对于编译文件夹，使用 **builddir**，对于安装文件夹，使用 **installdir**。

```
cd "matlabroot\extern\engines\python"
python setup.py build --build-base="builddir" install --prefix="installdir"
```

在您的主文件夹中安装引擎

要安装仅供您自己使用的引擎 API，请使用 **--user** 选项安装到您的主文件夹中。

```
cd "matlabroot\extern\engines\python"
python setup.py install --user
```

当您使用 **--user** 进行安装时，不需要将您的主文件夹添加到 **PYTHONPATH**。

另请参阅

相关示例

- “安装用于 Python 的 MATLAB Engine API” （第 13-4 页）

详细信息

- “用于 Python 的 MATLAB 引擎 API 的系统要求” （第 13-35 页）

启动和停止用于 Python 的 MATLAB 引擎

启动用于 Python 的 MATLAB 引擎

- 在操作系统提示符下启动 Python。
- 将 **matlab.engine** 包导入您的 Python 会话中。
- 通过调用 **start_matlab** 启动新的 MATLAB 进程。**start_matlab** 函数返回 Python 对象 **eng**，您可以通过该对象传递数据和调用由 MATLAB 执行的函数。

```
import matlab.engine
eng = matlab.engine.start_matlab()
```

使用启动选项启动引擎

启动引擎，并将选项作为输入参数字符串传递给 **matlab.engine.start_matlab**。例如，随桌面启动 MATLAB。

```
eng = matlab.engine.start_matlab("-desktop")
```

您可以使用单个字符串定义多个启动选项。例如，启动桌面并将数值显示格式设置为 **short**。

```
eng = matlab.engine.start_matlab("-desktop -r 'format short'")
```

您也可以在启动引擎后启动桌面。

```
import matlab.engine
eng = matlab.engine.start_matlab()
eng.desktop(nargout=0)
```

启动特定 MATLAB 引擎版本

要启动特定版本的 MATLAB 引擎，请将 **PYTHONPATH** 环境变量设置为包的位置。以下代码假设您使用了上一节中显示的设置。要将 Windows 上的 **PYTHONPATH** 设置为调用 MATLAB R2022a，请键入：

```
sys.path.append("c:\work\matlab22aPy39")
```

在 Linux 或 macOS 上：

```
sys.path.append("/local/work/matlab22aPy39")
```

要检查导入了哪个版本的 MATLAB，请在 Python 中键入：

```
import matlab
print(matlab.__file__)
```

Python 可能使用不同文件夹名称进行安装。例如，Python 在安装 MATLAB 引擎之前可能会创建一个子文件夹 **lib/site-packages**。请验证系统上的文件夹，以便与 **sys.path.append** 命令结合使用。

异步启动引擎

以异步方式启动引擎。在 MATLAB 启动时，您可以在 Python 命令行中输入命令。

```
import matlab.engine
future = matlab.engine.start_matlab(background=True)
```

创建 MATLAB 实例，以便在 MATLAB 中执行计算。

```
eng = future.result()
```

运行多个引擎

分别启动每个引擎。每个引擎启动自己的 MATLAB 进程并与之通信。

```
eng1 = matlab.engine.start_matlab()
eng2 = matlab.engine.start_matlab()
```

停止引擎

调用 **exit** 或 **quit** 函数。

```
eng.quit()
```

如果在引擎仍在运行的情况下退出 Python，Python 会自动停止引擎及其 MATLAB 进程。

另请参阅

matlab.engine.start_matlab

详细信息

- “指定启动选项”
- “常用启动选项”
- “通过 Python 调用 MATLAB 函数” （第 13-11 页）

将 Python 连接到正在运行的 MATLAB 会话

您可以将用于 Python 的 MATLAB 引擎连接到已在您的本地机器上运行的共享 MATLAB 会话。您也可以从单一 Python 会话连接到多个共享 MATLAB 会话。您可以在 MATLAB 会话期间的任何时间共享该会话，也可以在使用启动选项启动该会话时共享它。

连接到共享 MATLAB 会话

首先，将您的 MATLAB 会话转换为共享会话。从 MATLAB 调用 **matlab.engine.shareEngine**。

```
matlab.engine.shareEngine
```

在操作系统提示符下启动 Python。要连接到共享 MATLAB 会话，请从 Python 中调用 **matlab.engine.connect_matlab**。您可以从 Python 中调用任何 MATLAB 函数。

```
import matlab.engine
eng = matlab.engine.connect_matlab()
eng.sqrt(4.0)
```

```
2.0
```

您可以按名称连接到共享会话。要查找共享会话的名称，请从 Python 调用 **matlab.engine.find_matlab**。

```
matlab.engine.find_matlab()
```

```
('MATLAB_13232',)
```

matlab.engine.find_matlab 返回一个 **tuple**，其中包含您的本地机器上所有共享 MATLAB 会话的名称。在本示例中，**matlab.engine.shareEngine** 为共享会话提供了默认名称 **MATLAB_13232**，其中 13232 是 MATLAB 进程的 ID。每当您启动 MATLAB 时，操作系统都会为 MATLAB 会话提供一个不同的进程 ID。

按名称连接到 MATLAB 会话。

```
eng.quit()
newEngine = matlab.engine.connect_matlab('MATLAB_13232')
```

如果您未指定具体共享会话的名称，则 **matlab.engine.connect_matlab** 会连接到由 **matlab.engine.find_matlab** 返回的 **tuple** 中指定的第一个会话。

异步连接到共享 MATLAB 会话

从 MATLAB 中，将您的 MATLAB 会话转换为共享会话。

```
matlab.engine.shareEngine
```

在操作系统提示符下启动 Python。异步连接到共享 MATLAB 会话。

```
import matlab.engine
future = matlab.engine.connect_matlab(background=True)
eng = future.result()
```

从 Python 调用 MATLAB 函数。

```
eng.sqrt(4.0)
```

2.0

连接到多个共享 MATLAB 会话

您可以从 Python 连接到多个共享 MATLAB 会话。

启动另一个 MATLAB 会话。从 MATLAB 调用 **matlab.engine.shareEngine**。为第二个共享会话命名。该名称必须是有效的 MATLAB 变量名称。有关有效变量名称的信息，请参阅“变量名称”。

```
matlab.engine.shareEngine('MATLABEngine2')
```

从 Python 中查找所有共享 MATLAB 会话。

```
import matlab.engine
matlab.engine.find_matlab()
```

```
('MATLAB_13232','MATLABEngine2')
```

要连接到共享 MATLAB 会话，请从 Python 中调用 **matlab.engine.connect_matlab**。

```
eng1 = matlab.engine.connect_matlab('MATLAB_13232')
eng2 = matlab.engine.connect_matlab('MATLABEngine2')
```

使用启动选项启动共享 MATLAB 会话

默认情况下，MATLAB 会话不共享。但是，您可以使用启动选项将 MATLAB 作为共享会话启动。

在操作系统提示符下启动共享 MATLAB 会话。

```
matlab -r "matlab.engine.shareEngine"
matlab -r "matlab.engine.shareEngine('MATLABEngine3')"
```

您可以使用默认名称启动会话，或者用单引号括起名称来启动会话。

另请参阅

matlab.engine.shareEngine | **matlab.engine.isEngineShared** | **matlab.engine.engineName** | **matlab.engine.connect_matlab** | **matlab.engine.find_matlab**

详细信息

- “指定启动选项”
- “常用启动选项”

通过 Python 调用 MATLAB 函数

使用 MATLAB Engine API for Python 调用 MATLAB 路径中的任何 MATLAB 函数。

如果 MATLAB 函数不在路径中，您可以从当前文件夹中调用它。例如，要调用文件夹 **myFolder** 中的 MATLAB 函数 **myFnc**，请键入：

```
import matlab.engine
eng = matlab.engine.start_matlab()
eng.cd(r'myFolder', nargout=0)
eng.myFnc()
```

从 MATLAB 函数返回输出参数

您可以直接调用任何 MATLAB 函数并将结果返回到 Python。例如，要确定某个数是否为质数，请使用该引擎调用 **isprime** 函数。

```
import matlab.engine
eng = matlab.engine.start_matlab()
tf = eng.isprime(37)
print(tf)
```

```
True
```

从 MATLAB 函数返回多个输出参数

当使用引擎调用函数时，默认情况下该引擎会返回单个输出参数。如果您知道函数可能返回多个参数，请使用 **nargout** 参数指定输出参数的数量。

要确定两个数的最大公分母，请使用 **gcd** 函数。设置 **nargout** 以从 **gcd** 返回三个输出参数。

```
import matlab.engine
eng = matlab.engine.start_matlab()
t = eng.gcd(100.0,80.0,nargout=3)
print(t)
```

```
(20.0, 1.0, -1.0)
```

不从 MATLAB 函数返回任何输出参数

有些 MATLAB 函数不会返回任何输出参数。如果函数不返回任何参数，则将 **nargout** 设为 0。

通过 Python 打开 MATLAB 帮助浏览器。

```
import matlab.engine
eng = matlab.engine.start_matlab()
eng.doc(nargout=0)
```

MATLAB **doc** 函数将打开浏览器，但不会返回输出参数。如果您没有指定 **nargout=0**，引擎将报告错误。

停止执行函数

要停止执行 MATLAB 函数，请按 **Ctrl+C**。控制权将返回给 Python。

用函数名称替代 MATLAB 运算符

您可以通过调用等效函数在 Python 中使用 MATLAB 运算符。有关运算符和关联的函数名称的列表，请参阅 “MATLAB 运算符和关联的函数”。例如，要添加两个数值，请使用 **plus** 函数，而不是 + 运算符。

```
import matlab.engine
eng = matlab.engine.start_matlab()
a = 2
b = 3
eng.plus(a,b)
```

另请参阅

matlab.engine.MatlabEngine | matlab.engine.FutureResult

相关示例

- “从 Python 以异步方式调用 MATLAB 函数” （第 13-13 页）
- “从 Python 中调用用户脚本和函数” （第 13-14 页）
- “在 Python 中使用 MATLAB 数组” （第 13-28 页）
- “从 Python 对 MATLAB 数据进行分类并绘图” （第 13-29 页）

从 Python 以异步方式调用 MATLAB 函数

此示例说明如何从 Python 异步调用 MATLAB **sqrt** 函数，并稍后检索平方根。

默认情况下，引擎同步调用 MATLAB 函数。仅当 MATLAB 函数完成时，控制权才会返还给 Python。但是，引擎也可以异步调用函数。当 MATLAB 仍在执行该函数时，控制权会立即返还给 Python。引擎将结果存储在 Python 变量中，可以在函数完成后检查该变量。

使用 **background** 参数异步调用 MATLAB 函数。

```
import matlab.engine
eng = matlab.engine.start_matlab()
future = eng.sqrt(4.0,background=True)
ret = future.result()
print(ret)
```

```
2.0
```

使用 **done** 方法检查异步调用是否完成。

```
tf = future.done()
print(tf)
```

```
True
```

要在函数完成前停止执行，请调用 **future.cancel()**。

另请参阅

matlab.engine.MatlabEngine | matlab.engine.FutureResult

相关示例

- “通过 Python 调用 MATLAB 函数” （第 13-11 页）
- “从 Python 中调用用户脚本和函数” （第 13-14 页）

从 Python 中调用用户脚本和函数

此示例显示如何通过 Python 来调用 MATLAB 脚本，以计算三角形的面积。

要调用 MATLAB 脚本或函数，请将其放在您的 MATLAB 路径中。对于此示例，在当前文件夹中名为 **triarea.m** 的文件中创建一个 MATLAB 脚本。

```
b = 5;
h = 3;
a = 0.5*(b.* h)
```

保存该文件后，启动 Python 并调用该脚本。

```
import matlab.engine
eng = matlab.engine.start_matlab()
eng.triarea(nargout=0)
```

```
a =

   7.5000
```

指定 **nargout=0**。尽管脚本会打印输出，但它不会向 Python 返回任何输出参数。

将脚本转换为函数并通过引擎调用该函数。要编辑文件，请打开 MATLAB 编辑器。

```
eng.edit('triarea',nargout=0)
```

删除三个语句。然后添加一条函数声明并保存文件。

```
function a = triarea(b,h)
a = 0.5*(b.* h);
```

通过引擎调用新的 **triarea** 函数。

```
ret = eng.triarea(1.0,5.0)
print(ret)
```

```
2.5
```

triarea 函数仅返回一个输出参数，因此无需指定 **nargout**。

另请参阅

matlab.engine.MatlabEngine | matlab.engine.FutureResult

相关示例

- “通过 Python 调用 MATLAB 函数” （第 13-11 页）

将标准输出和错误重定向到 Python

此示例说明如何将标准输出和标准错误从 MATLAB 函数重定向到 Python **StringIO** 对象。

使用 **io** 模块创建 **StringIO** 对象。

```
import matlab.engine
eng = matlab.engine.start_matlab()
import io
out = io.StringIO()
err = io.StringIO()
ret = eng.dec2base(2**60,16,stdout=out,stderr=err)
```

当输入参数大于 2^52 时，**dec2base** 引发异常。显示在 **err** 中捕获的错误消息。

```
print(err.getvalue())
```

```
Error using dec2base (line 22)
First argument must be an array of integers, 0 <= D <= 2^52.
```

另请参阅

matlab.engine.MatlabEngine | matlab.engine.FutureResult

相关示例

- “通过 Python 调用 MATLAB 函数” （第 13-11 页）

在 Python 中使用 MATLAB 句柄对象

此示例说明如何从 MATLAB 句柄类创建对象，并在 Python 中调用其方法。

在当前文件夹中，在名为 **Triangle.m** 的文件中创建一个 MATLAB 句柄类。

```
classdef Triangle < handle
   properties (SetAccess = private)
      Base = 0;
      Height = 0;
   end

   methods
      function TR = Triangle(b,h)
         TR.Base = b;
         TR.Height = h;
      end

      function a = area(TR)
         a = 0.5 .* TR.Base .* TR.Height;
      end

      function setBase(TR,b)
         TR.Base = b;
      end

      function setHeight(TR,h)
         TR.Height = h;
      end
   end
end
```

启动 Python。创建一个 **Triangle** 句柄对象，并使用引擎调用其 **area** 方法。将句柄对象作为第一个位置参数传递。

```
import matlab.engine
eng = matlab.engine.start_matlab()
tr = eng.Triangle(5.0,3.0)
a = eng.area(tr)
print(a)
```

```
7.5
```

将 **tr** 复制到 MATLAB 工作区。您可以使用 **eval** 从工作区访问句柄对象的属性。

```
eng.workspace["wtr"] = tr
b = eng.eval("wtr.Base")
print(b)
```

```
5.0
```

使用 **setHeight** 方法更改高度。如果您的 MATLAB 句柄类定义了属性的 get 和 set 方法，则无需使用 MATLAB 工作区即可访问属性。

```
eng.setHeight(tr,8.0,nargout=0)
a = eng.area(tr)
print(a)
```

20.0

注意 **Triangle** 类对象 **tr** 是对象的句柄，而不是对象的副本。如果是在函数中创建的 **tr**，则它仅在该函数的作用域内有效。

另请参阅

matlab.engine.MatlabEngine | matlab.engine.FutureResult

相关示例

- “通过 Python 调用 MATLAB 函数” (第 13-11 页)

在 Python 中使用 MATLAB 引擎工作区

此示例说明如何在 Python 中将变量添加到 MATLAB 引擎工作区。

当您启动引擎时，它提供与所有 MATLAB 变量的集合的一个接口。此集合名为 **workspace**，它被实现为附加到引擎的 Python 字典。每个 MATLAB 变量的名称都成为 **workspace** 字典中的一个键。**workspace** 中的键必须是有效的 MATLAB 标识符（例如，您不能将数字用作键）。您可以在 Python 中将变量添加到引擎工作区，然后即可在 MATLAB 函数中使用这些变量。

将变量添加到引擎工作区。

```
import matlab.engine
eng = matlab.engine.start_matlab()
x = 4.0
eng.workspace['y'] = x
a = eng.eval('sqrt(y)')
print(a)
```

```
2.0
```

在本示例中，**x** 仅作为 Python 变量存在。其值被赋给引擎工作区中的一个新条目 **y**，从而创建一个 MATLAB 变量。然后，您可以调用 MATLAB **eval** 函数以在 MATLAB 中执行 **sqrt(y)** 语句并将输出值 2.0 返回到 Python。

另请参阅

matlab.engine.MatlabEngine | **matlab.engine.FutureResult**

相关示例

- “通过 Python 调用 MATLAB 函数” （第 13-11 页）
- “从 Python 对 MATLAB 数据进行分类并绘图” （第 13-29 页）

从 Python 将数据传递到 MATLAB

Python 类型到 MATLAB 标量类型的映射

当您将 Python 数据作为输入参数传递到 MATLAB 函数时，MATLAB Engine for Python 会将数据转换为等效的 MATLAB 数据类型。

Python 输入参数类型 - 仅标量值	生成的 MATLAB 数据类型
float	**double**
complex	复数 **double**
int	**int64**
float(nan)	**NaN**
float(inf)	**Inf**
bool	**logical**
str	**char**
dict	如果所有键都是字符串，则为结构体 否则不支持

Python 容器到 MATLAB 数组类型的映射

Python 输入参数类型 - 容器	生成的 MATLAB 数据类型
matlab 数值数组对象（请参阅“MATLAB 数组作为 Python 变量” （第 13-23 页））	数值数组
bytearray	**uint8** 数组
bytes	**uint8** 数组
list	元胞数组
set	元胞数组
tuple	元胞数组

不支持的 Python 类型

MATLAB Engine API 不支持以下 Python 类型。

- Python 类 (**module.type**) 对象
- **None**

另请参阅

详细信息

- “处理从 MATLAB 返回到 Python 的数据” （第 13-21 页）

- “MATLAB 和 Python 中的默认数值类型” (第 13-34 页)
- “MATLAB 数组作为 Python 变量” (第 13-23 页)

处理从 MATLAB 返回到 Python 的数据

MATLAB 标量类型到 Python 类型的映射

当 MATLAB 函数返回输出参数时，用于 Python 的 MATLAB 引擎 API 会将数据转换为等同的 Python 数据类型。

MATLAB 输入参数类型 - 仅标量值	生成的 Python 数据类型
double	**float**
single	**float**
复数（任意数值类型）	**complex**
int8	**int**
uint8	**int**
int16	**int**
uint16	**int**
int32	**int**
uint32	**int** (Python 3.x) **long** (Python 2.7)
int64	**int** (Python 3.x) **long** (Python 2.7)
uint64	**int** (Python 3.x) **long** (Python 2.7)
NaN	**float(nan)**
Inf	**float(inf)**
logical	**bool**
string	**string**
string 中的 **<missing>** 值	**None**
返回到 Python 3.x 的 **char**	**str**
返回到 Python 2.7 的 **char**	**str**（当 MATLAB **char** 值小于或等于 127 时） **unicode**（当 MATLAB **char** 值大于 127 时）
结构体	**dict**
MATLAB 句柄对象（例如 **containers.Map** 类型）	**matlab.object** MATLAB 返回对 **matlab.object** 的引用，而不是返回对象本身。您不能在 MATLAB 会话之间传递 **matlab.object**。
MATLAB 值对象（例如 **categorical** 类型）	不透明对象。您可以将值对象传递给 MATLAB 函数，但不能创建或修改它。

MATLAB 数组类型到 Python 类型的映射

MATLAB 输出参数类型 - 数组	生成的 Python 数据类型
数值数组	**matlab** 数值数组对象（请参阅 “MATLAB 数组作为 Python 变量” （第 13-23 页））
string 向量	**string** 的 **list**
返回到 Python 3.x 的 **char** 数组（1×**N**、**N**×1）	**str**
返回到 Python 2.7 的 **char** 数组（1×**N**、**N**×1）	**str**（当 MATLAB **char** 数组的值小于或等于 127 时） **unicode**（当 MATLAB **char** 数组的值大于 127 时）
行或列元胞数组	**list**

不支持的 MATLAB 类型

用于 Python 的 MATLAB 引擎 API 不支持以下 MATLAB 数据类型。

- **char** 数组（**M**×**N**）
- 元胞数组（**M**×**N**）
- 稀疏数组
- 结构体数组
- 非 MATLAB 对象（例如 Java 对象）

另请参阅

详细信息

- “从 Python 将数据传递到 MATLAB” （第 13-19 页）

MATLAB 数组作为 Python 变量

matlab Python 模块提供数组类以将 MATLAB 数值类型的数组表示为 Python 变量，以便 MATLAB 数组可以在 Python 和 MATLAB 之间传递。

matlab.engine Python 模块中的 MATLAB 类

- 通过导入 **matlab.engine** Python 包并调用必要的构造函数，您可以在 Python 代码中使用 MATLAB 数值数组。例如：

  ```
  import matlab.engine
  a = matlab.double([[1, 2, 3],[4, 5, 6]])
  ```

  构造函数的名称表示 MATLAB 数值类型。您可以将 MATLAB 数组作为输入参数传递给从 Python 调用的 MATLAB 函数。当 MATLAB 函数将数值数组作为输出参数返回时，该数组返回到 Python。
- 您可以使用包含数字的可选 **initializer** 输入参数初始化数组。**initializer** 参数必须为 Python 序列类型，如 **list**、**tuple** 或 **range**。您可以指定 **initializer** 包含多个数字序列。
- 您可以使用可选的 **vector** 输入参数（该参数包含大小为 1×N 的输入）初始化数组。如果使用 **vector**，则无法使用 **initializer**。
- 您可以使用以下选项之一创建一个多维数组：
  - 指定一个嵌套序列，但不指定大小。
  - 指定一个嵌套序列，并指定与该嵌套序列的维度匹配的 **size** 输入参数。
  - 指定一个一维序列和一个多维大小。在本例中，将假设序列以列优先顺序表示元素。
- 通过将可选的 **is_complex** 关键字参数设置为 **True**，您可以创建复数 MATLAB 数组。
- 您可以在 Python 中使用自定义类型初始化 MATLAB 数组。自定义类型应实现 Python 缓冲区协议。NumPy 中的 **ndarray** 就是一个示例。

matlab Python 包中的类	Python 中的构造函数调用
matlab.double	**matlab.double(initializer=None|vector=None, size=None, is_complex=False)**
matlab.single	**matlab.single(initializer=None|vector=None, size=None, is_complex=False)**
matlab.int8	**matlab.int8(initializer=None|vector=None, size=None, is_complex=False)**
matlab.int16	**matlab.int16(initializer=None|vector=None, size=None, is_complex=False)**

matlab Python 包中的类	Python 中的构造函数调用
matlab.int32	matlab.int32(initializer=None|vector=None, size=None, is_complex=False)
matlab.int64	matlab.int64(initializer=None|vector=None, size=None, is_complex=False)
matlab.uint8	matlab.uint8(initializer=None|vector=None, size=None, is_complex=False)
matlab.uint16	matlab.uint16(initializer=None|vector=None, size=None, is_complex=False)
matlab.uint32	matlab.uint32(initializer=None|vector=None, size=None, is_complex=False)
matlab.uint64	matlab.uint64(initializer=None|vector=None, size=None, is_complex=False)
matlab.logical	matlab.logical(initializer=None|vector=None, size=None)[a]

a Logicals cannot be made into an array of complex numbers.

matlab Python 包中 MATLAB 类的属性和方法

使用 **matlab.engine** 包构造函数创建的所有 MATLAB 数组都具有以下属性和方法：

属性

属性名称	描述	示例

size	表示数组维数的整数组成的元组	>>> a = matlab.int16([[1, 2, 3],[4, 5, 6]])
		>>> a.size
		(2, 3)

itemsize	表示数组元素的字节大小的整数	>>> a = matlab.int16()
		>>> a.itemsize
		2
		>>> b = matlab.int32()
		>>> b.itemsize
		4

方法

方法名称	用途	示例

clone()	返回一个新的不同对象，其内容与原始对象的内容相同	**>>> a = matlab.int16(
		[[1, 2, 3],[4, 5, 6]])
		>>> b = a.clone()
		>>> print(b)
		[[1,2,3],[4,5,6]]
		>>> b[0][0] = 100
		>>> b matlab.int16(
		[[100,2,3],[4,5,6]])
		>>> print(a)
		[[1,2,3],[4,5,6]]**

real()	以 1×N 数组形式按列优先顺序返回复数元素的实部	**>>> a = matlab.int16([[1 + 10j, 2 + 20j, 3 + 30j],[4, 5, 6]], is_complex=True) >>> print(a.real()) [1,4,2,5,3,6]**
imag()	以 1×N 数组形式按列优先顺序返回复数元素的虚部	**>>> a = matlab.int16([[1 + 10j, 2 + 20j, 3 + 30j],[4, 5, 6]], is_complex=True) >>> print(a.imag()) [10,0,20,0,30,0]**
noncomplex()	以 1×N 数组形式按列优先顺序返回非复数元素	**>>> a = matlab.int16([[1, 2, 3],[4, 5, 6]]) >>> print(a.noncomplex()) [1,4,2,5,3,6]**

• **reshape(dim1,dim2,...,dimN)**
• **reshape((dim1,dim2,...,dimN))**
• **reshape([dim1,dim2,...,dimN])**
	根据维数重构数组并返回结果	**>>> a = matlab.int16(
		[[1, 2, 3],[4, 5, 6]])
		>>> print(a)
		[[1,2,3],[4,5,6]]
		>>> a.reshape(3, 2)
		>>> print(a)
		[[1,5],[4,3],[2,6]]**

toarray()	返回基于内容构造的标准 Python **array.array** 对象。仅适用于一维序列。	**>>> a = matlab.int16(
		[[1, 2, 3],[4, 5, 6]])
		>>> a[0].toarray()
		array('h', [1, 2, 3])
		>>> b = matlab.int16(
		[[1 + 10j, 2 + 20j,
		3 + 30j],[4, 5, 6]],
		is_complex=True)
		>>> b.real().toarray()
		array('h', [1, 4, 2, 5, 3, 6])**

tomemoryview()	返回基于内容构造的标准 Python **memoryview** 对象	**>>> a = matlab.int16(
		[[1, 2, 3],[4, 5, 6]])
		>>> b = a.tomemoryview()
		>>> b.tolist()
		[[1, 2, 3], [4, 5, 6]]
		>>> b.shape
		(2, 3)**

Python 中的多维 MATLAB 数组

在 Python 中，您可以创建任何数值类型的多维 MATLAB 数组。使用两个 Python **list** 变量创建一个 2×5 MATLAB 双精度数组。

```
import matlab.engine
A = matlab.double([[1,2,3,4,5], [6,7,8,9,10]])
print(A)
```

```
[[1.0,2.0,3.0,4.0,5.0],[6.0,7.0,8.0,9.0,10.0]]
```

A 的 **size** 属性显示它是 2×5 数组。

```
print(A.size)
```

```
(2, 5)
```

在 Python 中对 MATLAB 数组进行索引

就像您可以对 Python **list** 和 **tuple** 变量进行索引一样，您也可以对 MATLAB 数组进行索引。

```
import matlab.engine
A = matlab.int8([1,2,3,4,5])
print(A[0])
```

```
[1,2,3,4,5]
```

MATLAB 数组的大小为 **(1,5)**；因此，**A[0]** 是 **[1,2,3,4,5]**。对该数组进行索引会得到 3。

```
print(A[0][2])
```

```
3
```

Python 索引是从 0 开始的。当在 Python 会话中访问 MATLAB 数组的元素时，请使用从 0 开始的索引。

对多维 MATLAB 数组进行索引。

```
A = matlab.double([[1,2,3,4,5], [6,7,8,9,10]])
print(A[1][2])
```

```
8.0
```

在 Python 中对 MATLAB 数组进行切片

您可以像对 Python **list** 和 **tuple** 变量进行切片一样，对 MATLAB 数组进行切片。

```
import matlab.engine
A = matlab.int8([[1,2,3,4,5]])
print(A[0][1:4])
```

```
[2,3,4]
```

您可以将数据分配到一个切片。以下代码显示从 Python **list** 到一个数组切片的分配。

```
A = matlab.double([[1,2,3,4],[5,6,7,8]]);
A[0] = [10,20,30,40]
print(A)
```

```
[[10.0,20.0,30.0,40.0],[5.0,6.0,7.0,8.0]]
```

您可以分配来自另一个 MATLAB 数组或来自包含数字的任何 Python 可迭代对象的数据。

您可以为分配指定切片，如下所示。

```
A = matlab.int8([1,2,3,4,5,6,7,8]);
A[0][2:4] = [30,40]
A[0][6:8] = [70,80]
print(A)
```

```
[[1,2,30,40,5,6,70,80]]
```

在 Python 中重构 MATLAB 数组

您可以使用 **reshape** 方法在 Python 中重构 MATLAB 数组。输入参数 **size** 必须是保留元素数量的一个序列。使用 **reshape** 将 1×9 MATLAB 数组更改为 3×3。

```
import matlab.engine
A = matlab.int8([1,2,3,4,5,6,7,8,9])
A.reshape((3,3))
print(A)
```

```
[[1,4,7],[2,5,8],[3,6,9]]
```

另请参阅

相关示例

- “在 Python 中使用 MATLAB 数组” （第 13-28 页）

在 Python 中使用 MATLAB 数组

此示例说明如何在 Python 中创建 MATLAB 数组并将其作为输入参数传递给 MATLAB **sqrt** 函数。

matlab 包提供了构造函数以支持在 Python 中创建 MATLAB 数组。用于 Python 的 MATLAB 引擎 API 可以将此类数组作为输入参数传递给 MATLAB 函数，并且可以将此类数组作为输出参数返回给 Python。您可以从 Python 序列类型创建任何 MATLAB 数值或逻辑值类型的数组。

从 Python **list** 创建一个 MATLAB 数组。对该数组调用 **sqrt** 函数。

```
import matlab.engine
eng = matlab.engine.start_matlab()
a = matlab.double([1,4,9,16,25])
b = eng.sqrt(a)
print(b)
```

```
[[1.0,2.0,3.0,4.0,5.0]]
```

引擎返回 **b**，它是 1×5 的 **matlab.double** 数组。

创建一个多维数组。**magic** 函数将一个二维 **matlab.double** 数组返回给 Python。使用 **for** 循环分行打印数组中的每行。（当看到 **...** 提示时再次按 **Enter** 以关闭循环并打印。）

```
a = eng.magic(6)
for x in a: print(x)
...
```

```
[35.0,1.0,6.0,26.0,19.0,24.0]
[3.0,32.0,7.0,21.0,23.0,25.0]
[31.0,9.0,2.0,22.0,27.0,20.0]
[8.0,28.0,33.0,17.0,10.0,15.0]
[30.0,5.0,34.0,12.0,14.0,16.0]
[4.0,36.0,29.0,13.0,18.0,11.0]
```

调用 **tril** 函数来获取 **a** 的下三角部分。在一个单独的行上打印数组中的每行。

```
b = eng.tril(a)
for x in b: print(x)
...
```

```
[35.0,0.0,0.0,0.0,0.0,0.0]
[3.0,32.0,0.0,0.0,0.0,0.0]
[31.0,9.0,2.0,0.0,0.0,0.0]
[8.0,28.0,33.0,17.0,0.0,0.0]
[30.0,5.0,34.0,12.0,14.0,0.0]
[4.0,36.0,29.0,13.0,18.0,11.0]
```

另请参阅

相关示例

- “通过 Python 调用 MATLAB 函数” （第 13-11 页）

详细信息

- “MATLAB 数组作为 Python 变量” （第 13-23 页）

从 Python 对 MATLAB 数据进行分类并绘图

此示例说明如何在 Python 中将患者数据分类到吸烟者和非吸烟者列表中，并对 MATLAB 患者的血压读数绘图。

启动引擎，将一组患者的数据读入 MATLAB 表。MATLAB 提供以逗号分隔的示例文件 **patients.dat**，其中包含 100 名不同患者的信息。

```
import matlab.engine
eng = matlab.engine.start_matlab()
eng.eval("T = readtable('patients.dat');",nargout=0)
```

MATLAB **readtable** 函数将数据读入表中。引擎不支持 MATLAB 表数据类型。不过，使用 MATLAB **table2struct** 函数，您可以将表转换为标量结构体，这是引擎支持的数据类型。

```
eng.eval("S = table2struct(T,'ToScalar',true);",nargout=0)
eng.eval("disp(S)",nargout=0)
```

```
             LastName: {100x1 cell}
               Gender: {100x1 cell}
                  Age: [100x1 double]
             Location: {100x1 cell}
               Height: [100x1 double]
               Weight: [100x1 double]
               Smoker: [100x1 double]
             Systolic: [100x1 double]
            Diastolic: [100x1 double]
    SelfAssessedHealthStatus: {100x1 cell}
```

您可以将 MATLAB 工作区中的 **S** 传递给您的 Python 会话。引擎将 **S** 转换为 Python 字典 **D**。

```
D = eng.workspace["S"]
```

S 的字段包含数组。该引擎将元胞数组转换为 Python **list** 变量，并将数值数组转换为 MATLAB 数组。因此，**D["LastName"]** 的数据类型为 **list**，而 **D["Age"]** 的数据类型为 **matlab.double**。

将血压读数分为吸烟者和非吸烟者列表。在 **patients.dat** 中，**Smoker** 列用逻辑值 1 (true) 表示吸烟者，用逻辑值 0 (false) 表示非吸烟者。将 **D["Smoker"]** 转换为 **matlab.logical** 数组以进行分类。

```
smoker = matlab.logical(D["Smoker"])
```

将 **Diastolic** 血压读数和 **Smoker** 指示符转换为 1×100 MATLAB 数组以进行分类。

```
pressure = D["Diastolic"]
pressure.reshape((1,100))
pressure = pressure[0]
smoker.reshape((1,100))
smoker = smoker[0]
```

将 **pressure** 数组分为吸烟者和非吸烟者的血压读数列表。Python 列表推导式为迭代序列提供了一种紧凑的方法。使用 Python **zip** 函数，可以在单个 **for** 循环中迭代多个序列。

```
sp = [p for (p,s) in zip(pressure,smoker) if s is True]
nsp = [p for (p,s) in zip(pressure,smoker) if s is False]
```

显示吸烟者的血压读数 **sp**（**list** 类型）的长度。

```
print(len(sp))
```

```
34
```

显示非吸烟者读数 **nsp**（**list** 类型）的长度。

```
print(len(nsp))
```

```
66
```

计算吸烟者和非吸烟者的平均血压读数。将 **sp** 和 **nsp** 转换为 MATLAB 数组，然后将它们传递给 MATLAB **mean** 函数。

```
sp = matlab.double(sp)
nsp = matlab.double(nsp)
print(eng.mean(sp))
```

```
89.9117647059
```

显示非吸烟者的均值血压。

```
print(eng.mean(nsp))
```

```
79.3787878788
```

绘制吸烟者和非吸烟者的血压读数。要为绘图定义两个 x 轴，请调用 MATLAB **linspace** 函数。您可以在同一个散点图上绘制 34 个吸烟者和 66 个非吸烟者。

```
sdx = eng.linspace(1.0,34.0,34)
nsdx = eng.linspace(1.0,34.0,66)
```

使用 **box** 函数显示轴边界。

```
eng.figure(nargout=0)
eng.hold("on",nargout=0)
eng.box("on",nargout=0)
```

您必须使用 **nargout=0** 调用 **figure**、**hold** 和 **box** 函数，因为这些函数不返回输出参数。

绘制吸烟者和非吸烟者的血压读数，并对绘图加标签。对于许多 MATLAB 函数，引擎可以返回 MATLAB 图形对象的句柄。您可以将 MATLAB 对象的句柄存储在 Python 变量中，但无法操作 Python 中的对象属性。您可以将 MATLAB 对象作为输入参数传递给其他 MATLAB 函数。

```
eng.scatter(sdx,sp,10,'blue')
```

```
<matlab.object object at 0x22d1510>
```

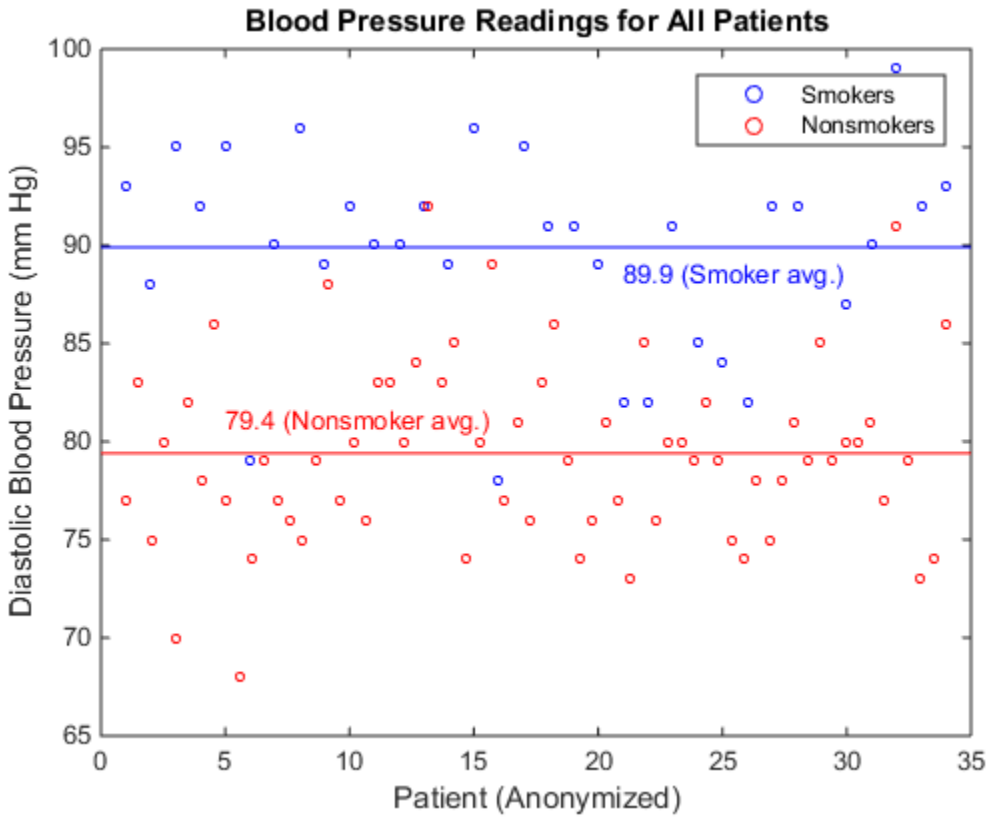
在此示例的其余部分，将 MATLAB 函数的输出参数赋给占位符 **h**。

```
h = eng.scatter(nsdx,nsp,10,'red')
h = eng.xlabel("Patient (Anonymized)")
h = eng.ylabel("Diastolic Blood Pressure (mm Hg)")
h = eng.title("Blood Pressure Readings for All Patients")
h = eng.legend("Smokers","Nonsmokers")
```

绘制吸烟者和非吸烟者的平均血压读数。

```
x = matlab.double([0,35])
y = matlab.double([89.9,89.9])
h = eng.line(x,y,"Color","blue")
h = eng.text(21.0,88.5,"89.9 (Smoker avg.)","Color","blue")
```

```
y = matlab.double([79.4,79.4])
h = eng.line(x,y,"Color","red")
h = eng.text(5.0,81.0,"79.4 (Nonsmoker avg.)","Color","red")
```



另请参阅

readtable | **scatter**

从 Python 获取 MATLAB 函数的帮助

如何查找 MATLAB 帮助

您可以从 Python 访问所有 MATLAB 函数的支持文档。此文档包括示例，并说明每个函数的输入参数、输出参数和调用语法。

用于 Python 的 MATLAB 引擎 API 支持您使用 MATLAB 的 **doc** 和 **help** 函数。使用 **doc** 打开 MATLAB 帮助浏览器。在 Python 提示符下，使用 **help** 获取 MATLAB 函数的简要说明。

从 Python 打开 MATLAB 帮助浏览器

您可以从 Python 使用帮助浏览器打开 MATLAB 函数参考页并搜索文档。

例如，显示 MATLAB **plot** 函数的参考页。（由于 **doc** 不返回输出参数，您必须设置 **nargout=0**。）

```
import matlab.engine
eng = matlab.engine.start_matlab()
eng.doc("plot",nargout=0)
```

参考页包括函数说明、示例和相关文档的链接。

注意 请点击示例标题，如果 MATLAB 参考页上未显示示例，请点击标题旁边的箭头。示例可以在页内折叠或展开。

如果不带位置参数调用 **eng.doc**，它将打开帮助浏览器。（您仍须设置关键字参数 **nargout=0**。）

```
eng.doc(nargout=0)
```

要搜索 MATLAB 文档，请在帮助浏览器中任意页顶部的搜索框中键入表达式。浏览器返回搜索结果列表，突出显示与表达式匹配的字词。

您也可以使用 **docsearch** 函数搜索文档。例如，搜索提及 **plot** 的页。

```
eng.docsearch("plot",nargout=0)
```

在 Python 提示符下显示 MATLAB 帮助

要在 Python 提示符下显示函数的帮助文本，请调用 MATLAB **help** 函数。例如，显示 **erf** 的帮助文本。

```
import matlab.engine
eng = matlab.engine.start_matlab()
eng.help("erf",nargout=0)
```

```
 ERF Error function.
    Y = ERF(X) is the error function for each element of X.  X must be
    real. The error function is defined as:

      erf(x) = 2/sqrt(pi) * integral from 0 to x of exp(-t^2) dt.

    See also ERFC, ERFCX, ERFINV, ERFCINV.

    Other functions named erf:
```

```
    codistributed/erf
    gpuArray/erf
    sym/erf

  Reference page in Help browser
    doc erf
```

输出显示帮助文本，但不包括任何可能提到的其他 MATLAB 函数的帮助链接。

MATLAB 和 Python 中的默认数值类型

默认情况下，MATLAB 以双精度浮点数形式存储所有数值。而 Python 默认情况下将一些数值存储为整数。由于这种差异，您可能会将整数作为输入参数传递给需要双精度数值的 MATLAB 函数。

以 MATLAB 中的这些变量赋值为例：

```
x = 4;
y = 4.0;
```

x 和 **y** 的数据类型均为 **double**。现在，相同的赋值在 Python 中如下所示：

```
x = 4
y = 4.0
```

x 和 **y** 具有不同的数值数据类型。

```
print(type(x))
```

```
<type 'int'>
```

```
print(type(y))
```

```
<type 'float'>
```

大多数 MATLAB 函数接受数据类型为 **double** 的数值输入参数。最佳做法是确保作为输入参数传递给 MATLAB 函数的数值具有 Python 数据类型 **float**，而不是 Python 数据类型 **int**。如果您要进行以下操作，请确保 Python 变量是浮点数：

- 字面值使用浮点数。例如，键入 **4.0**，而不是 **4**。
- 转换为数据类型 **float**。例如，**x = float(4)** 将数值转换为数据类型 **float**。
- 从数值或序列创建 **matlab.double** 数组。例如，**x = matlab.double([1,2,3,4,5])** 从 Python 整数 **list** 创建具有 **double** 数据类型的 MATLAB 数组。

当您将整数传递给接受数据类型为 **double** 的输入参数的 MATLAB 函数时，引擎会引发错误。有关示例，请参阅 “MatlabExecutionError: Undefined Function”。

当调用接受整数作为数值输入参数的 MATLAB 函数时，可以将具有 Python 数据类型 **int** 的输入参数传递给该函数。

用于 Python 的 MATLAB 引擎 API 的系统要求

您可以在 MATLAB 支持的任何平台上使用用于 Python 的 MATLAB 引擎 API。

Python 版本支持

要使用适用于 Python 的 MATLAB 引擎 API，您必须在您的系统上安装受支持的 Python 参考实现（也称为 CPython）版本。有关支持的版本信息，请参阅 MATLAB 产品（按版本）兼容的 Python 版本。

要下载并安装 Python，请参阅“安装支持的 Python 实现” （第 21-10 页）。

注意 要在 Microsoft Windows 系统上安装 64 位 MATLAB 的 2.7 版本，请选择 64 位 Python 版本，称为 Windows x86-64 MSI installer。

要从操作系统提示符下调用 Python，请执行以下操作之一。

- 将 Python 的完整路径添加到 PATH 环境变量中
- 调用 Python 解释器时包括完整路径

要确定您是否在调用支持的版本，请在操作系统提示符下键入 `python -V` 以显示 Python 版本号。

有关 Python 语言的帮助，请参阅 `python.org` 文档中的 `www.python.org/doc`。如需关于第三方模块或用户定义模块的帮助，请参考产品文档。

下载 Python 和 MATLAB 的 64 位版本

MATLAB 的架构必须与 Python 的架构匹配。在 Python 下载网站上，针对 Microsoft Windows 平台的下载默认为 32 位版本。要下载 64 位版本，请选择名称为 Windows x86-64 MSI installer 的选项。

要测试您的 Python 版本是 32 位还是 64 位，请在 Python 提示符下键入以下代码：

```
import sys
print(sys.maxsize > 2**32)
```

如果 Python 解释器是 64 位，则以下代码返回 `True`；如果是 32 位，则返回 `False`。（有关详细信息，请参阅 Python 2.7 Documentation — Cross Platform。）

从源文件编译 Python 的要求

要在 Linux 上启用对 Python 2.7 的 wide-unicode 支持，请使用 `--enable-unicode=ucs4` 选项配置编译。在 Mac 系统上编译任何版本的 Python 或在 Linux 上编译 Python 3.x 时，不需要此配置选项。

另请参阅

详细信息

- “安装支持的 Python 实现” （第 21-10 页）
- MATLAB 产品（按版本）兼容的 Python 版本

外部网站

- www.python.org/doc

MATLAB Engine API for Python 的限制

用于 Python 的 MATLAB 引擎 API 不支持以下功能。

- 引擎无法启动或连接到在远程计算机上的 MATLAB。
- Python 关键字参数不能作为使用引擎调用的 MATLAB 函数的输入参数。引擎只将位置参数传递给 MATLAB 函数。
- 在 Python 和 MATLAB 之间传递的数据数组的大小限制为不超过 2 GB。此限制应用于进程间传递的数据和支持信息。
- 递归数据结构体不能作为输入参数传递给 MATLAB 函数，也无法放入引擎工作区中。（递归数据结构体是将自身作为一个值包含在内的 Python 数据结构体。）

MATLAB Engine API 不支持以下 Python 类型。

- Python 类 (**module.type**) 对象
- **None**

另请参阅

详细信息

- “Troubleshoot MATLAB Errors in Python”
- MATLAB 产品（按版本）兼容的 Python 版本

Engine API for C++

编译 C++ Engine 程序的要求

支持的编译器

使用支持 C++11 的编译器。有关支持的编译器的最新列表，请参阅支持和兼容的编译器。

使用 mex 命令进行编译

您可以使用 MATLAB 编辑器编写您的引擎应用程序代码，并运行 **mex** 命令编译它。要为 C++ Engine 应用程序设置编译器，请键入：

```
mex -setup -client engine C++
```

要编译您的 C++ Engine 程序 **MyEngineCode.cpp**，请键入：

```
mex -client engine MyEngineCode.cpp
```

要测试您的设置，请参阅“Test Your C++ Build Environment”。

使用 IDE 进行编译

要使用集成开发环境 (IDE)（如 Microsoft Visual Studio 或 Xcode）编写源代码，请使用以下库和包含文件来设置编译 C++ Engine 应用程序的环境。引擎应用程序需要引擎库 **libMatlabEngine**、MATLAB 数据数组库 **libMatlabDataArray** 和支持包含文件。

在以下路径设定中，将 **matlabroot** 替换为 MATLAB **matlabroot** 命令返回的路径。

Windows 库

在这些路径设定中，将 **compiler** 替换为 **microsoft** 或 **mingw64**。

- 引擎库 - **matlabroot\extern\lib\win64\compiler\libMatlabEngine.lib**
- MATLAB 数据数组库 - **matlabroot\extern\lib\win64\compiler\libMatlabDataArray.lib**

macOS 库

- 引擎库 - **matlabroot/extern/bin/maci64/libMatlabEngine.dylib**
- MATLAB 数据数组库 - **matlabroot/extern/bin/maci64/libMatlabDataArray.dylib**

Linux 库

- 引擎库 - **matlabroot/extern/bin/glnxa64/libMatlabEngine.so**
- MATLAB 数据数组库 - **matlabroot/extern/bin/glnxa64/libMatlabDataArray.so**

其他库 - **pthread**

例如，要编译 myEngineApp.cpp，请使用下列库。将 **matlabroot** 替换为 MATLAB **matlabroot** 命令返回的路径。

```
g++ -std=c++11 -I matlabroot/extern/include/ -L matlabroot/extern/bin/glnxa64/
   -pthread myEngineApp.cpp -lMatlabDataArray -lMatlabEngine
```

引擎 Include 文件

头文件包含函数声明以及您在 API 库中访问的例程的原型。这些文件位于 **matlabroot/extern/include** 文件夹中，并且与用于 Windows、macOS 和 Linux 系统的文件相同。引擎应用程序使用：

- **MatlabEngine.hpp** - C++ Engine API 的定义
- **MatlabDataArray.hpp** - C++ 和 MATLAB 数据之间的泛型接口的定义

MATLAB 数据数组是类和 API 的集合，提供外部数据和 MATLAB 之间的泛型接口。

运行时环境

要运行您的应用程序，请将以下环境变量之一设置为指定的路径。

操作系统	变量	路径
Windows	**PATH**	**matlabroot\extern\bin\win64**
macOS	**DYLD_LIBRARY_PATH**	**matlabroot/extern/bin/maci64**
Linux	**LD_LIBRARY_PATH**	**matlabroot/extern/bin/glnxa64:matlabroot/sys/os/glnxa64**

另请参阅

mex | **matlab::engine::MATLABEngine**

相关示例

- “C++ Engine API”
- “Test Your C++ Build Environment”

从 C++ 调用 MATLAB 函数

使用 **matlab::engine::MATLABEngine** 类的 “feval” 和 “fevalAsync” 成员函数从 C++ 调用 MATLAB 函数。当您要将函数参数从 C++ 传递给 MATLAB 和将函数执行的结果返回给 C++ 时，请使用这些函数。这些成员函数的工作方式类似于 MATLAB **feval** 函数。

要调用 MATLAB 函数，需要满足以下条件：

- 将函数名称作为 **matlab::engine::String** 传递。
- 定义 MATLAB 函数所需的输入参数。您可以使用原生 C++ 数据类型或 MATLAB 数据 API。有关详细信息，请参阅“MATLAB Data API for C++”。
- 指定 MATLAB 函数应提供的输出的数目。默认为一个输出。有关详细信息，请参阅“使用多个返回参数调用函数” （第 14-7 页）和“控制输出的数目” （第 14-8 页）。
- 为 MATLAB 函数的结果定义适当的返回类型。
- 使用流缓冲区将标准输出和标准错误从 MATLAB 命令行窗口重定向到 C++。有关详细信息，请参阅“Redirect MATLAB Command Window Output to C++”

要使用 MATLAB 基础工作区中的变量计算 MATLAB 语句，请使用 **matlab::engine::MATLABEngine** “eval” 和 “evalAsync” 成员函数。这些函数使您能够在 MATLAB 工作区中创建和使用变量，但不返回值。有关详细信息，请参阅“Evaluate MATLAB Statements from C++”。

有关如何设置和编译 C++ 引擎程序的信息，请参阅“编译 C++ Engine 程序的要求” （第 14-2 页）。

调用带单一返回参数的函数

此示例使用 MATLAB **gcd** 函数求两个数值的最大公约数。**MATLABEngine::feval** 成员函数返回 **gcd** 函数调用的结果。

使用 **matlab::data::ArrayFactory** 创建两个标量 **int16_t** 参数。将参数以 **std::vector** 形式传递给 **MATLABEngine::feval**。

```
#include "MatlabEngine.hpp"
#include "MatlabDataArray.hpp"
#include <iostream>

void callFevalgcd() {

    // Pass vector containing MATLAB data array scalar
    using namespace matlab::engine;

    // Start MATLAB engine synchronously
    std::unique_ptr<MATLABEngine> matlabPtr = startMATLAB();

    // Create MATLAB data array factory
    matlab::data::ArrayFactory factory;

    // Pass vector containing 2 scalar args in vector
    std::vector<matlab::data::Array> args({
        factory.createScalar<int16_t>(30),
        factory.createScalar<int16_t>(56) });

    // Call MATLAB function and return result
    matlab::data::TypedArray<int16_t> result = matlabPtr->feval(u"gcd", args);
    int16_t v = result[0];
    std::cout << "Result: " << v << std::endl;
}
```

您可以使用原生 C++ 类型调用 **MATLABEngine::feval**。为此，您必须将调用 **MATLABEngine::feval** 的返回类型指定为：

feval<type>(...)

例如，此处返回类型是 **int**：

```
int cresult = matlabPtr->feval<int>(u"gcd", 30, 56);
```

此示例定义一个 **matlab::data::TypedArray**，以将 **double** 类型的数组传递给 MATLAB **sqrt** 函数。由于数组中的数值之一是负数，因此 MATLAB 返回复数数组作为结果。因此，将返回类型定义为 **matlab::data::TypedArray<std::complex<double>>**。

```
#include "MatlabDataArray.hpp"
#include "MatlabEngine.hpp"
#include <iostream>

void callFevalsqrt() {
   // Call MATLAB sqrt function on array

   using namespace matlab::engine;

   // Start MATLAB engine synchronously
   std::unique_ptr<MATLABEngine> matlabPtr = startMATLAB();

   // Create  MATLAB data array factory
   matlab::data::ArrayFactory factory;

   // Define a four-element array
   matlab::data::TypedArray<double> const argArray =
      factory.createArray({ 1,4 }, { -2.0, 2.0, 6.0, 8.0 });

   // Call MATLAB function
   matlab::data::TypedArray<std::complex<double>> const results =
      matlabPtr->feval(u"sqrt", argArray);

   // Display results
   int i = 0;
   for (auto r : results) {
      double a = argArray[i++];
      double realPart = r.real();
      double imgPart = r.imag();
      std::cout << "Square root of " << a << " is " <<
         realPart << " + " << imgPart << "i" << std::endl;
   }
}
```

在调用 MATLAB 函数时，对返回类型使用 **matlab::data::Array** 是安全的。例如，您可以对返回值使用 **matlab::data::Array** 来编写前面的示例。

```
void callFevalsqrt() {
   // Call MATLAB sqrt function on array

   using namespace matlab::engine;

   // Start MATLAB engine synchronously
   std::unique_ptr<MATLABEngine> matlabPtr = startMATLAB();

   // Create MATLAB data array factory
   matlab::data::ArrayFactory factory;

   // Define a four-element array
   matlab::data::Array const argArray =
      factory.createArray({ 1,4 }, { -2.0, 2.0, 6.0, 8.0 });

   // Call MATLAB function
   matlab::data::Array results = matlabPtr->feval(u"sqrt", argArray);

   // Display results
   for (int i = 0; i < results.getNumberOfElements(); i++) {
      double a = argArray[i];
      std::complex<double> v = results[i];
      double realPart = v.real();
      double imgPart = v.imag();
      std::cout << "Square root of " << a << " is " <<
         realPart << " + " << imgPart << std::endl;
   }
}
```

使用名称/值参数调用函数

一些 MATLAB 函数接受可选的名称-值对组参数。名称是字符数组，值可以是任何类型的值。使用 **std::vector** 创建包含正确序列的名称和值的参数向量。

此示例代码调用 MATLAB **movsum** 函数来计算行向量的三点中心移动和，而放弃端点计算。此函数调用需要以下参数：

- 数值数组
- 标量窗长度
- 名称-值对组由字符数组 **Endpoint** 和 **discard** 组成

以下是等效的 MATLAB 代码：

```
A = [4 8 6 -1 -2 -3 -1 3 4 5];
M = movsum(A,3,'Endpoints','discard');
```

将这些要用于 MATLAB 函数的参数以 **std::vector** 形式传递给 **MATLABEngine::feval**。使用 **matlab::data::ArrayFactory** 创建每个参数。

```
void callFevalmovsum() {
   //Pass vector containing various types of arguments

   using namespace matlab::engine;

   // Start MATLAB engine synchronously
   std::unique_ptr<MATLABEngine> matlabPtr = startMATLAB();

   // Create  MATLAB data array factory
   matlab::data::ArrayFactory factory;

  // Create a vector of input arguments
   std::vector<matlab::data::Array> args({
      factory.createArray<double>({ 1, 10 }, { 4, 8, 6, -1, -2, -3, -1, 3, 4, 5 }),
      factory.createScalar<int32_t>(3),
      factory.createCharArray("Endpoints"),
      factory.createCharArray("discard")
   });

   // Call MATLAB function
   matlab::data::TypedArray<double> const result = matlabPtr->feval(u"movsum", args);

   // Display results
   int i = 0;
   for (auto r : result) {
      std::cout << "results[" << i++ << "] = " << r << std::endl;
   }
}
```

以异步方式调用函数

此示例调用 MATLAB **conv** 函数来将两个多项式相乘。在调用 **MATLABEngine::fevalAsync** 后，使用 **FutureResult::get** 从 MATLAB 获得结果。

```
#include "MatlabDataArray.hpp"
#include "MatlabEngine.hpp"
#include <iostream>

static void callFevalAsync() {
   //Call MATLAB functions asynchronously

   using namespace matlab::engine;

   // Start MATLAB engine synchronously
   std::unique_ptr<MATLABEngine> matlabPtr = startMATLAB();

   // Create MATLAB data array factory
```

```
    matlab::data::ArrayFactory factory;

     // Create input argument arrays
    std::vector<matlab::data::Array> args({
        factory.createArray<double>({ 1, 3 },{ 1, 0, 1 }),
        factory.createArray<double>({ 1, 2 },{ 2, 7 })
    });
    String func(u"conv");

    // Call function asnychronously
    FutureResult<matlab::data::Array> future = matlabPtr->fevalAsync(func, args);

    // Get results
    matlab::data::TypedArray<double> results = future.get();

    // Display results
    std::cout << "Coefficients: " << std::endl;
    for (auto r : results) {
        std::cout << r << " " << std::endl;
    }
}
```

使用多个返回参数调用函数

以下示例代码使用 MATLAB **gcd** 函数对传递的两个数值输入求最大公约数和 Bézout 系数。**gcd** 函数可以返回一个或三个参数，具体取决于函数调用请求的输出的数目。在此示例中，对 MATLAB **gcd** 函数的调用返回三个输出。

默认情况下，**MATLABEngine::feval** 假设返回值的数目为 1。因此，您必须将返回值的实际数目指定为 **MATLABEngine::feval** 的第二个参数。

在此示例中，**MATLABEngine::feval** 返回一个 **std::vector**，其中包含 **gcd** 函数调用的三个结果。返回值是整数标量。

```
#include "MatlabDataArray.hpp"
#include "MatlabEngine.hpp"
#include <iostream>

void multiOutput() {
    //Pass vector containing MATLAB data array array

    using namespace matlab::engine;

    // Start MATLAB engine synchronously
    std::unique_ptr<MATLABEngine> matlabPtr = startMATLAB();
    std::cout << "Started MATLAB Engine" << std::endl;

    //Create MATLAB data array factory
    matlab::data::ArrayFactory factory;

    //Create vector of MATLAB data array arrays
    std::vector<matlab::data::Array> args({
        factory.createScalar<int16_t>(30),
        factory.createScalar<int16_t>(56)
    });

    //Call gcd function, get 3 outputs
    const size_t numReturned = 3;
    std::vector<matlab::data::Array> result = matlabPtr->feval(u"gcd", numReturned, args);

    //Display results
    for (auto r : result) {
        std::cout << "gcd output: " << int16_t(r[0]) << std::endl;
    }
}
```

用原生 C++ 类型调用函数

调用 MATLAB 函数时，可以使用原生 C++ 类型。**MATLABEngine::feval** 和 **MATLABEngine::fevalAsync** 接受作为 MATLAB 函数参数传递的某些标量 C++ 类型。要将数组和其

他类型传递给 MATLAB 函数，请使用 MATLAB 数据 API。有关此 API 的详细信息，请参阅 “MATLAB Data API for C++” 。

此示例使用 **int16_t** 值作为输入，使用 **std::tuple** 从 MATLAB **gcd** 函数返回结果。

以下是等效的 MATLAB 代码。

```
[G,U,V] = gcd(int16(30),int16(56));
```

```
#include "MatlabEngine.hpp"
#include <iostream>
#include <tuple>

void multiOutputTuple() {
   //Return tuple from MATLAB function call

   using namespace matlab::engine;

   // Start MATLAB engine synchronously
   std::unique_ptr<MATLABEngine> matlabPtr = startMATLAB();

   //Call MATLAB gcd function
   std::tuple<int16_t, int16_t, int16_t> nresults;
   nresults = matlabPtr->feval<std::tuple<int16_t, int16_t, int16_t>>
      (u"gcd", int16_t(30), int16_t(56));

   // Display results
   int16_t G;
   int16_t U;
   int16_t V;
   std::tie(G, U, V) = nresults;
   std::cout << "GCD : " << G << ", "
           << "Bezout U: " << U << ", "
           << "Bezout V: " << V << std::endl;
}
```

有关成员函数语法的具体信息，请参阅 **matlab::engine::MATLABEngine**。

控制输出的数目

根据请求的输出数量，MATLAB 函数的行为可能会有所不同。某些函数不返回任何输出或返回指定数量的输出。

例如，MATLAB **pause** 函数使执行暂停指定的秒数。但是，如果您使用一个输出参数调用 **pause**，它将立即返回状态值而不存在暂停。

```
pause(20) % Pause for 20 seconds

state = pause(20); % No pause, return pause state
```

此示例调用 **pause** 但不指定输出。指定 **void** 输出后，MATLAB 暂停执行 20 秒。

```
#include "MatlabEngine.hpp"

void voidOutput() {
   // No output from feval
   using namespace matlab::engine;

   // Start MATLAB engine synchronously
   std::unique_ptr<MATLABEngine> matlabPtr = startMATLAB();

   // Call pause function with no output
   matlabPtr->feval<void>(u"pause", 20);
}
```

对 **MATLABEngine::feval** 的以下调用使用将 MATLAB 函数参数定义为 **std::vector<matlab::data::Array>** 的签名。在没有指定输出参数的情况下，MATLAB 将暂停执行 20 秒。

```
#include "MatlabDataArray.hpp"
#include "MatlabEngine.hpp"

void zeroOutput() {
   // No output from feval

   using namespace matlab::engine;

   // Start MATLAB engine synchronously
   std::unique_ptr<MATLABEngine> matlabPtr = startMATLAB();

   //Create MATLAB data array factory
   matlab::data::ArrayFactory factory;

   // Call pause function with no output
   matlab::data::Array arg = factory.createScalar<int16_t>(20);
   const size_t numReturned = 0;
   matlabPtr->feval(u"pause", numReturned, { arg });
}
```

另请参阅

matlab::data::ArrayFactory | **matlab::engine::MATLABEngine**

相关示例

- “Evaluate MATLAB Statements from C++”

通过 MATLAB 使用 .NET 库

读取 Excel 电子表格数据的元胞数组

此示例适用于 Microsoft .NET Framework，说明如何将 Microsoft Excel® 电子表格数据的列转换为 MATLAB 类型。MATLAB 将一系列 .NET 值作为 **System.Object[,]** 类型进行读取。使用 **cell** 函数将 **System.String** 值转换为 MATLAB 字符数组，将 **System.DateTime** 值转换为 **datetime** 对象。

在 Excel 中创建包含以下数据的文件。

```
Date      Weight
10/31/96  174.8
11/29/96  179.3
12/30/96  190.4
01/31/97  185.7
```

右键点击 **Date** 列，选择**设置单元格格式**，然后选择**数字**选项卡。请验证“分类”的值为 **Date**。

命名文件夹 **H:\Documents\MATLAB** 中的 **weight.xls** 文件。关闭文件。

在 MATLAB 中，从电子表格中读取数据。

```
dotnetenv("framework")
NET.addAssembly('microsoft.office.interop.excel');
app = Microsoft.Office.Interop.Excel.ApplicationClass;
book =  app.Workbooks.Open('H:\Documents\MATLAB\weight.xls');
sheet = Microsoft.Office.Interop.Excel.Worksheet(book.Worksheets.Item(1));
range = sheet.UsedRange;
arr = range.Value;
```

将数据转换为 MATLAB 类型。

```
data = cell(arr,'ConvertTypes',{'all'});
```

显示日期。

```
cellfun(@disp,data(:,1))
```

```
Date
  31-Oct-1996 00:00:00

  29-Nov-1996 00:00:00

  30-Dec-1996 00:00:00

  31-Jan-1997 00:00:00
```

退出 Excel 程序。

```
Close(book)
Quit(app)
```

另请参阅

相关示例

- “使用 Excel 作为自动化服务器读取电子表格数据” （第 17-2 页）

详细信息

- “Convert .NET Arrays to Cell Arrays”
- “电子表格”

在 MATLAB 函数中使用 import

如果您在 MATLAB 函数中使用 **import** 命令，请在调用该函数之前添加对应的 .NET 程序集。例如，以下函数 **getPrinterInfo** 调用 **System.Drawing** 命名空间中的方法。

```
function ptr = getPrinterInfo
import System.Drawing.Printing.*;
ptr = PrinterSettings;
end
```

要调用该函数，请键入：

```
dotnetenv("framework")
NET.addAssembly('System.Drawing');
printer = getPrinterInfo;
```

请不要将命令 **NET.addAssembly('System.Drawing')** 添加到 **getPrinterInfo** 函数中。MATLAB 会在执行 **NET.addAssembly** 命令之前处理 **getPrinterInfo.m** 代码。在这种情况下，**PrinterSettings** 不是完全限定的，MATLAB 不能识别该名称。

同样，**import** 命令的作用域仅限于 **getPrinterInfo** 函数。在命令行中，键入：

```
ptr = PrinterSettings;
```

```
Undefined function or variable 'PrinterSettings'.
```

另请参阅

import

创建泛型类型的 .NET 数组

此示例创建一个 .NET List<Int32> 泛型类型数组。

```
genType = NET.GenericClass('System.Collections.Generic.List',...
  'System.Int32');
arr = NET.createArray(genType, 5)
```

```
arr =

  List<System*Int32>[] with properties:

         Length: 5
     LongLength: 5
           Rank: 1
       SyncRoot: [1x1 System.Collections.Generic.List<System*Int32>[]]
     IsReadOnly: 0
    IsFixedSize: 1
   IsSynchronized: 0
```

16

Engine API for .NET

17

通过 MATLAB 使用 COM 对象

使用 Excel 作为自动化服务器读取电子表格数据

此示例说明如何使用 COM 自动化服务器从 MATLAB 访问另一个应用程序。该示例创建了用于访问 Microsoft Excel 文件中的数据的用户界面。如果您在应用程序中未使用组件对象模型 (COM)，请参阅“电子表格”中的函数和示例，以获取将 Excel 电子表格数据导入 MATLAB 的备选方法。

为实现 MATLAB 与电子表格程序之间的通信，该示例在运行 Excel 应用程序的自动化服务器中创建了一个对象。然后，MATLAB 通过 Excel 自动化服务器提供的接口访问电子表格中的数据。最后，该示例创建了用于访问 Microsoft Excel 文件中的数据的用户界面。

演示的方法

- 使用自动化服务器，从 MATLAB 访问其他应用程序。
- 将 Excel 数据转换为界面和绘图中所用类型的方法。

以下方法说明如何可视化和操作电子表格数据：

- 实现一个支持绘制 Excel 电子表格选定列的界面。
- 将 MATLAB 图窗插入 Excel 文件中。

要查看完整的代码列表，请在编辑器中打开文件 actx_excel.m。

创建 Excel 自动化服务器

通过 MATLAB 访问电子表格数据的第一步是使用 **actxserver** 函数和程序 ID **excel.application** 在自动化服务器进程中运行 Excel 应用程序。

exl 对象提供了对 Excel 程序所支持的多个接口的访问权限。使用 **Workbooks** 接口打开包含数据的 Excel 文件。

使用 workbook 的 **Sheets** 接口访问来自 **Range** 对象的数据，该对象存储了对指定工作表内某个数据范围的引用。此示例访问从列 **A** 中的第一个单元格到列 **G** 中的最后一个单元格的所有数据。

此时，将通过 range 对象接口 **rngObj** 访问来自 Excel 文件的 **sheet1** 的整个数据集。此对象将在 MATLAB 元胞数组 **exlData** 中返回数据，该元胞数组同时包含数值和字符数据：

操作 MATLAB 工作区中的数据

现在数据在元胞数组中，您可以使用 MATLAB 函数提取部分数据并将其重构，以在界面中使用和传递给绘图函数。有关数据的假设，请参阅 “Excel 电子表格格式” （第 17-3 页）。

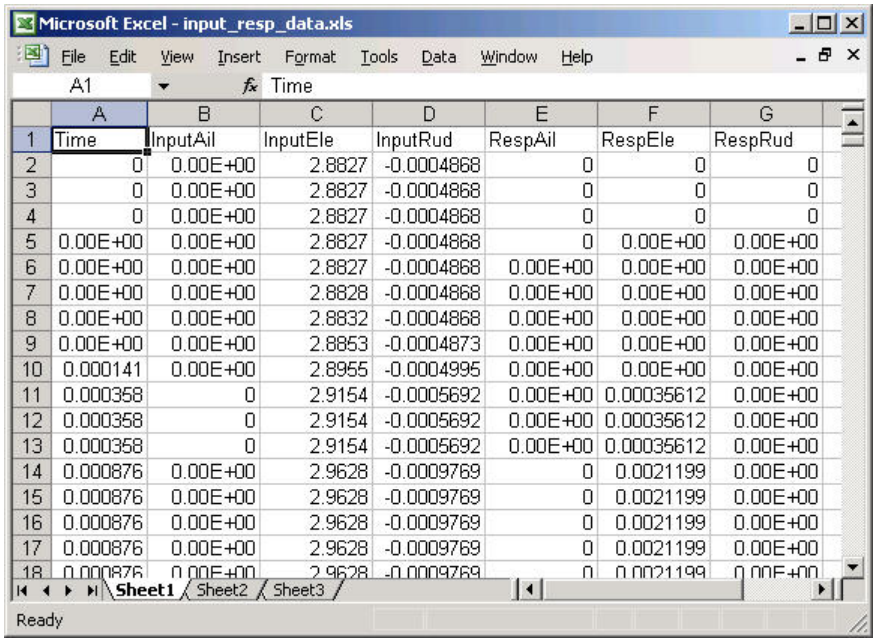
以下代码用于操作数据：

该代码执行以下操作：

- 从元胞数组中提取数值数据。请查看花括号 **{}** 内的索引表达式。
- 串联索引操作返回的各个双精度值。请查看方括号 **[]** 内的表达式。
- 使用 **reshape** 函数将结果重构为数组，该数组将数据排列在各个列中。
- 提取每一列 **exlData** 数据的第一个单元格中的文本，并将文本存储在元胞数组 **lBoxList** 中。此变量用于生成列表框中的项目。

Excel 电子表格格式

此示例假设 Excel 电子表格具有此图所示的特定组织方式。



	A	B	C	D	E	F	G
1	Time	InputAil	InputEle	InputRud	RespAil	RespEle	RespRud
2	0	0.00E+00	2.8827	-0.0004868	0	0	0
3	0	0.00E+00	2.8827	-0.0004868	0	0	0
4	0	0.00E+00	2.8827	-0.0004868	0	0	0
5	0.00E+00	0.00E+00	2.8827	-0.0004868	0	0.00E+00	0.00E+00
6	0.00E+00	0.00E+00	2.8827	-0.0004868	0.00E+00	0.00E+00	0.00E+00
7	0.00E+00	0.00E+00	2.8828	-0.0004868	0.00E+00	0.00E+00	0.00E+00
8	0.00E+00	0.00E+00	2.8832	-0.0004868	0.00E+00	0.00E+00	0.00E+00
9	0.00E+00	0.00E+00	2.8853	-0.0004873	0.00E+00	0.00E+00	0.00E+00
10	0.000141	0.00E+00	2.8955	-0.0004995	0.00E+00	0.00E+00	0.00E+00
11	0.000358	0	2.9154	-0.0005692	0.00E+00	0.00035612	0.00E+00
12	0.000358	0	2.9154	-0.0005692	0.00E+00	0.00035612	0.00E+00
13	0.000358	0	2.9154	-0.0005692	0.00E+00	0.00035612	0.00E+00
14	0.000876	0.00E+00	2.9628	-0.0009769	0	0.0021199	0.00E+00
15	0.000876	0.00E+00	2.9628	-0.0009769	0	0.0021199	0.00E+00
16	0.000876	0.00E+00	2.9628	-0.0009769	0	0.0021199	0.00E+00
17	0.000876	0.00E+00	2.9628	-0.0009769	0	0.0021199	0.00E+00
18	0.000876	0.00E+00	2.9628	-0.0009769	0	0.0021199	0.00E+00

该 Excel 文件的格式为：

- 每一列的第一个元素是用于标识该列中所含数据的文本。这些值被提取并用于填充列表框。
- 第一列 **Time** 用于其余所有数据绘图的 x 轴。
- 每个列中的所有行都将读入 MATLAB。

创建绘图函数界面

此示例使用一个界面，使您能够从输入和响应数据的列表中进行选择。所有数据都绘制为时间的函数，并且您可以继续向图中添加更多数据。添加到图形中的每个数据绘图都会使得图例也随之扩展。

该界面包括下列细节：

- 在您向图中添加数据时会更新的图例
- 用于从坐标区中清除所有图的清除按钮
- 用于将图保存为 PNG 文件并将其添加到另一个 Excel 文件的保存按钮
- 用于显示或隐藏当前所访问的 Excel 文件的切换按钮
- 用于终止自动化服务器的图窗删除功能

选择并绘制数据

当您点击 **Create Plot** 按钮时，其回调函数将查询列表框，以确定选中了哪些项目并绘制每个数据对时间的图。MATLAB 会更新图例以显示新数据，同时仍保留现有数据的图例。

清除坐标区

此绘图函数设计为当用户从列表框中选择数据时可持续添加图形。**Clear Graph** 按钮将清除并重置坐标区，并清除用于存储绘图数据标签（供图例使用）的变量。

显示或隐藏 Excel 文件

MATLAB 程序可访问在自动化服务器中运行的 Excel 应用程序的属性。通过将 `Visible` 属性设为 `1` 或 `0`，此回调可控制 Excel 文件的可见性。

关闭图窗并终止 Excel 自动化进程

由于 Excel 自动化服务器在 MATLAB 以外的单独进程中运行，因此您必须显式终止此进程。关闭界面后，没有必要再让此进程保持运行状态，因此该示例通过 `Quit` 方法，使用图窗的 `delete` 函数来终止 Excel 进程。您还需要终止用于保存图的 Excel 进程。有关终止此进程的信息，请参阅“将 MATLAB 图插入 Excel 电子表格” （第 17-4 页）。

将 MATLAB 图插入 Excel 电子表格

您可以在 Excel 文件中保存使用此界面创建的图。本示例使用一个单独的 Excel 自动化服务器进程来实现此目的。**Save Graph** 普通按钮的回调将创建图像并将其添加到 Excel 文件：

- 如屏幕中所见，坐标区和图例都被复制到一个配置为打印图形的不可见图窗（图窗的 `PaperPositionMode` 属性被设为 `auto`）。
- `print` 命令可创建 PNG 图像。
- 使用 `Shapes` 接口将图像插入 Excel 工作簿。

服务器和接口均在初始化阶段进行实例化：

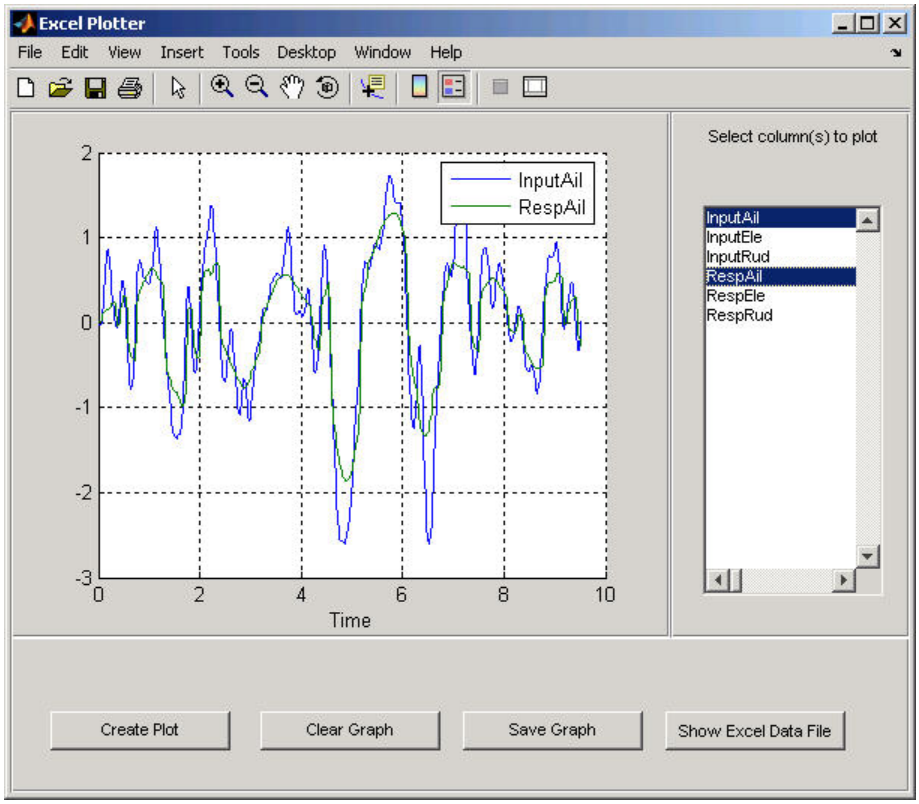
使用以下代码实现 **Save Graph** 按钮回调：

运行示例

要运行该示例，请选择列表框中的任意项目，然后点击 **Create Plot** 按钮。此示例提供的示例数据包含三个输入和三个关联的响应数据集。在该示例中，绘制所有这些数据集对 Excel 文件中的第一列（即时间数据）的图。

通过点击 **Show Excel Data File** 按钮查看 Excel 数据文件。要将图形图像保存在另一个 Excel 文件中，请点击 **Save Graph** 按钮。如果您在当前文件夹中拥有写访问权限，则 **Save Graph** 选项将在该文件夹中创建一个临时 PNG 文件。

界面如下图所示，列表框中的输入/响应对组为选中状态，坐标区则显示相应绘图。



要运行该示例，请点击此链接。

另请参阅

xlsread

详细信息

- “导入电子表格”

MATLAB COM 客户端支持

创建 COM 对象

使用 **actxserver** 函数，对动态链接库 (DLL) 组件创建一个进程内服务器，对可执行文件 (EXE) 组件创建一个进程外服务器。

实例化 DLL 组件

要为作为动态链接库 (DLL) 实现的组件创建一个服务器，请使用 **actxserver** 函数。MATLAB 在包含客户端应用程序的同一进程中创建组件实例。

当与 DLL 组件结合使用时，**actxserver** 的语法是 **actxserver(ProgID)**，其中 **ProgID** 是组件的编程标识符。

actxserver 返回对象的主接口的句柄。在其他 COM 函数调用中使用此句柄来引用该对象。您还可以使用该句柄来获取对象的更多接口。有关使用接口的详细信息，请参阅“COM Object Interfaces”。

与 Microsoft ActiveX® 控件不同，服务器显示的任何用户界面都显示在单独的窗口中。

您不能在 64 位 MATLAB 应用程序中使用 32 位进程内 DLL COM 对象。有关此限制的信息，请参阅 Why am I not able to use 32-bit DLL COM Objects in 64-bit MATLAB?。

实例化 EXE 组件

您可以使用 **actxserver** 函数为实现为可执行文件 (EXE) 的组件创建一个服务器。在这种情况下，MATLAB 在进程外服务器中实例化组件。

用于创建可执行文件的 **actxserver** 的语法是 **actxserver(ProgID, sysname)**。**ProgID** 是组件的编程标识符，**sysname** 是在配置分布式 COM (DCOM) 系统中使用的可选参数。

actxserver 返回 COM 对象的主接口的句柄。在其他 COM 函数调用中使用此句柄来引用该对象。您还可以使用该句柄来获取对象的更多接口。有关使用接口的详细信息，请参阅“COM Object Interfaces”。

服务器显示的任何用户界面都显示在单独的窗口中。

此示例创建一个运行 Microsoft Excel 电子表格程序的 COM 服务器应用程序。句柄被分配给 **h**。

```
h = actxserver('Excel.Application')
```

MATLAB 会转而显示以下结果：

```
h =
  COM.excel.application
```

MATLAB 可以以编程方式连接到已在计算机上运行的 COM 自动化服务器应用程序的实例。要获得对此类应用程序的引用，请使用 **actxGetRunningServer** 函数。

此示例获取对 Excel 程序的引用，该程序必须已在您的系统上运行。返回的句柄被分配给 **h**。

```
h = actxGetRunningServer('Excel.Application')
```

MATLAB 会转而显示以下结果：

```
h =
  COM.excel.application
```

另请参阅

actxserver | actxGetRunningServer

使用 Excel 作为自动化服务器写入电子表格数据

此示例显示如何将 MATLAB 矩阵写入 Excel 电子表格。有关将 MATLAB 数据导出到 Microsoft Excel 电子表格的备选方法，请参阅“电子表格”中的函数和示例。

创建 Excel 对象。

```
e = actxserver('Excel.Application');
```

添加一个工作簿。

```
eWorkbook = e.Workbooks.Add;
e.Visible = 1;
```

激活第一个工作表。

```
eSheets = e.ActiveWorkbook.Sheets;
eSheet1 = eSheets.get('Item',1);
eSheet1.Activate
```

将 MATLAB 数据置入工作表。

```
A = [1 2; 3 4];
eActivesheetRange = get(e.Activesheet,'Range','A1:B2');
eActivesheetRange.Value = A;
```

将数据读回 MATLAB，其中的数组 **B** 为元胞数组。

```
eRange = get(e.Activesheet,'Range','A1:B2');
B = eRange.Value;
```

将数据转换为双精度值矩阵。如果元胞数组仅包含标量值，则使用以下命令。

```
B = reshape([B{:}],size(B));
```

在文件中保存工作簿。

```
SaveAs(eWorkbook,'myfile.xlsx')
```

如果 Excel 程序显示关于保存文件的对话框，请选择相应的响应以继续。

如果已保存文件，则关闭工作簿。

```
eWorkbook.Saved = 1;
Close(eWorkbook)
```

退出 Excel 程序并删除服务器对象。

```
Quit(e)
delete(e)
```

注意 确保关闭您所创建的工作簿对象，以防止潜在的内存泄漏。

另请参阅

xlswrite

详细信息

- “Read Collection or Sequence of Spreadsheet Files”

连接到现有 Excel 应用程序

此示例说明如何在 MATLAB 中从打开的文件 **weekly_log.xlsx** 中读取数据。

MATLAB 可以访问由另一个应用程序打开的文件，方法是从 MATLAB 客户端创建一个 COM 服务器，然后通过此服务器打开该文件。

导航到一个包含 Excel 文件（例如 **weekly_log.xlsx**）的文件夹。在 Excel 程序中打开该文件。

在 MATLAB 中打开同一文件。

```
excelapp = actxserver('Excel.Application');
wkbk = excelapp.Workbooks;
wdata = Open(wkbk,'c:\work\weekly_log.xlsx');
```

从工作表 2 中读取 **D1** 到 **F6** 范围内的数据。

```
sheets = wdata.Sheets;
sheet12 = Item(sheets,2);
range = get(sheet12,'Range','D1','F6');
range.value
```

```
ans =

  'Temp.'     'Heat Index'    'Wind Chill'
  [78.4200]   [       32]    [       37]
  [69.7300]   [       27]    [       30]
  [77.6500]   [       17]    [       16]
  [74.2500]   [       -5]    [        0]
  [68.1900]   [       22]    [       35]
```

```
Close(wkbk)
Quit(excelapp)
```

另请参阅

actxserver

19

MATLAB COM 自动化服务器支持

将 MATLAB 注册为 COM 服务器

何时注册 MATLAB

要将 MATLAB 用作 COM 服务器，您必须在 Windows 注册表中注册该应用程序。当您安装新版本的 MATLAB 时，MATLAB 会自动为所有用户将此版本注册为 COM 服务器。要查看注册了 MATLAB 的哪些版本，请启动 MATLAB 并键入：

```
comserver('query')
```

MATLAB 显示注册的 MATLAB 版本的安装路径。这些信息特定于您的配置，例如：

```
      User: 'C:\Program Files\MATLAB\R2020a\bin\win64\MATLAB.exe'
Administrator: 'C:\Program Files\MATLAB\R2019b\bin\win64\MATLAB.exe'
```

要了解本地用户帐户和管理特权以及 Windows 如何根据这些值选择 COM 服务器，请参考您的 Microsoft Windows 文档。

如果 MATLAB 的注册版本不是您的首选版本，请选择以下方法之一：

- “为当前用户注册 MATLAB” （第 19-2 页）
- “为所有用户注册 MATLAB” （第 19-3 页）
- “从操作系统提示符注册” （第 19-3 页）

为当前用户注册 MATLAB

如果您没有管理员特权，或您不使用管理员特权启动 MATLAB，您仍可以将 MATLAB 注册为 COM 服务器。

启动您要注册的 MATLAB 版本并使用 **comserver** 命令：

```
comserver('register')
```

此命令只为您的用户帐户注册 MATLAB。当您不使用管理特权启动您的 COM 应用程序时，应用程序将与此 MATLAB 版本进行通信。

要使用由管理员注册的 MATLAB 版本，请启动注册到您的用户帐户的 MATLAB 并使用 **comserver** 注销您的版本：

```
comserver('unregister')
comserver('query')
```

```
      User: ''
Administrator: 'C:\Program Files\MATLAB\R2019b\bin\win64\MATLAB.exe'
```

现在您的应用程序将与 MATLAB R2019b 通信。

注意 **comserver** 函数适用于 MATLAB R2020a 及更高版本。

为所有用户注册 MATLAB

您必须拥有管理员特权，才能为所有用户将 MATLAB 注册为 COM 服务器。根据您的用户帐户控制 (UAC) 设置，您可能需要右键点击 Windows 命令提示符或 MATLAB 图标并选择**以管理员身份运行**。如果该选项不可用，请与系统管理员联系。

如果您的系统上安装了多个版本的 MATLAB，则只能将一个版本注册为所有用户的默认版本。此版本的 MATLAB 会持续作为注册版本，直到您安装或注册了不同版本的 MATLAB。

启动您要注册的 MATLAB 版本并使用 **comserver** 命令：

```
comserver('register','User','all')
```

注意 **comserver** 适用于 MATLAB R2020a 及更高版本。要注册以前版本的 MATLAB，请调用 **regmatlabserver** 函数。

从操作系统提示符注册

要从 Windows 系统提示符将 MATLAB 注册为 COM 服务器，请先使用**以管理员身份运行**选项打开 Windows 命令提示符。

使用以下命令转至包含要注册的 MATLAB 版本的可执行文件的文件夹：

```
cd matlabroot\bin\win64
```

其中 **matlabroot** 是 MATLAB 安装文件夹的完整路径。在 MATLAB 中调用 **matlabroot** 以获取该值。如果不使用此文件夹，则 **matlab** 命令会启动系统路径上的第一个 MATLAB 实例。

要注册 MATLAB，请执行以下命令：

```
matlab -batch "comserver('register','User', 'all')"
```

MATLAB 显示最小化的命令行窗口。打开此窗口并退出 MATLAB。

注意 **comserver** 适用于 MATLAB R2020a 及更高版本。要注册以前版本的 MATLAB，请使用 **matlab -regserver** 选项。

注销 MATLAB 作为 COM 服务器

有关如何以及何时注销 MATLAB 的信息，请参阅 **comserver**。

另请参阅

matlab (Windows) | **comserver** | **regmatlabserver**

详细信息

- “Create MATLAB Server”

从 Visual Basic .NET 客户端调用 MATLAB 函数

注意 要在 MATLAB R2022b 或更高版本中从 Microsoft Visual Basic .NET 应用程序调用 MATLAB 函数，请考虑使用 MATLAB Engine API for .NET。有关详细信息，请参阅“从 .NET 调用 MATLAB”。有关示例，请参阅 **MathWorks.MATLAB.Engine.MATLABEngine**。

如果您需要维护为 MATLAB R2022a 或更早版本创建的现有程序，请使用此示例，它通过 COM 接口从 Visual Basic .NET 客户端应用程序调用 MATLAB 函数。此示例还在新 MATLAB 窗口中绘制图形并执行简单的计算。

Visual Basic .NET 程序

```
Dim MatLab As Object
Dim Result As String
Dim MReal(1, 3) As Double
Dim MImag(1, 3) As Double

MatLab = CreateObject("Matlab.Application")

'Call MATLAB function from VB
Result = MatLab.Execute("surf(peaks)")

'Execute simple computation
Result = MatLab.Execute("a = [1 2 3 4; 5 6 7 8]")
Result = MatLab.Execute("b = a + a ")

'Bring matrix b into VB program
MatLab.GetFullMatrix("b", "base", MReal, MImag)
```

另请参阅

MathWorks.MATLAB.Engine.MATLABEngine | **GetFullMatrix** | **Execute**

详细信息

- “从 .NET 调用 MATLAB”
- “编写适用于 MATLAB 的 COM 应用程序”

将复数数据从 C# 客户端传递给 MATLAB

要将数据从 MATLAB R2022b 或更高版本中的 C# 应用程序传递给 MATLAB，请考虑使用 MATLAB Engine API for .NET。有关详细信息，请参阅 “Pass .NET Data Types to MATLAB Functions” 和 “Execute MATLAB Functions from .NET”。

如果您需要使用 MATLAB 作为 COM 自动化服务器来维护为 MATLAB R2022a 或更早版本创建的现有程序，请使用此示例，此示例在客户端 C# 程序中创建复数数据，并将其传递给 MATLAB。该矩阵由变量 **pr** 中的实数值向量和 **pi** 中的虚数值向量组成。该示例将矩阵读回 C# 程序中。

在 C# 中引用 MATLAB 类型库的语句如下：

```
MLApp.MLApp matlab = new MLApp.MLApp();
```

从 C# 客户端程序中，在您的项目中添加对 MATLAB COM 对象的引用。例如，在 Microsoft Visual Studio 中，打开您的项目。在**项目**菜单中，选择**添加引用**。在“添加引用”对话框中，选择 **COM** 选项卡。选择 MATLAB 应用程序。有关详细信息，请参考您的供应商文档。

以下是完整示例：

C# 程序

```
using System;
namespace ConsoleApplication4
{
class Class1
{
[STAThread]
static void Main(string[] args)
{
MLApp.MLApp matlab = new MLApp.MLApp();

System.Array pr = new double[4];
pr.SetValue(11,0);
pr.SetValue(12,1);
pr.SetValue(13,2);
pr.SetValue(14,3);

System.Array pi = new double[4];
pi.SetValue(1,0);
pi.SetValue(2,1);
pi.SetValue(3,2);
pi.SetValue(4,3);

matlab.PutFullMatrix("a", "base", pr, pi);

System.Array prresult = new double[4];
System.Array piresult = new double[4];

matlab.GetFullMatrix("a", "base", ref prresult, ref piresult);
}
}
}
```

另请参阅

GetFullMatrix | **PutFullMatrix**

详细信息

- “Pass .NET Data Types to MATLAB Functions”
- “Handle MATLAB Data in .NET Applications”

通过 C# 客户端调用 MATLAB 函数

注意 要在 MATLAB R2022b 或更高版本中从 C# 应用程序调用 MATLAB 函数，请考虑使用 MATLAB Engine API for .NET。有关详细信息，请参阅“从 .NET 调用 MATLAB”。有关示例，请参阅“Execute MATLAB Functions from .NET”。

如果您需要维护为 MATLAB R2022a 或更早版本创建的现有程序，请使用此示例，此示例说明如何使用 MATLAB 作为 COM 自动化服务器从 C# 应用程序调用用户定义的 MATLAB 函数 **myfunc**。该示例使用先前绑定的特定 MATLAB 版本。

注意 要使用此示例，您必须知道如何在开发环境（如 Microsoft Visual Studio）中创建和运行 COM 控制台应用程序。

在文件夹 **c:\temp\example** 中创建 MATLAB 函数 **myfunc**。

```
function [x,y] = myfunc(a,b,c)
x = a + b;
y = sprintf('Hello %s',c);
```

在您的开发环境中创建 C# 控制台应用程序。在 C# 中引用 MATLAB 类型库的语句如下：

```
MLApp.MLApp matlab = new MLApp.MLApp();
```

以下是完整示例：

C# 程序

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication2
{
  class Program
  {
    static void Main(string[] args)
    {
      // Create the MATLAB instance
      MLApp.MLApp matlab = new MLApp.MLApp();

      // Change to the directory where the function is located
      matlab.Execute(@"cd c:\temp\example");

      // Define the output
      object result = null;

      // Call the MATLAB function myfunc
      matlab.Feval("myfunc", 2, out result, 3.14, 42.0, "world");

      // Display result
      object[] res = result as object[];

      Console.WriteLine(res[0]);
```

```
            Console.WriteLine(res[1]);
            // Get user input to terminate program
            Console.ReadLine();
        }
    }
}
```

从 C# 客户端程序中，在您的项目中添加对 MATLAB COM 对象的引用。此引用将您的程序绑定到 MATLAB 的特定版本。有关详细信息，请参考您的供应商文档。例如，在 Microsoft Visual Studio 中，打开您的项目。在**项目**菜单中，选择**添加引用**。在“添加引用”对话框中，选择 **COM** 选项卡。选择 MATLAB 应用程序。

在开发环境中编译并运行该应用程序。

另请参阅

Execute | Feval (COM)

详细信息

- “Execute MATLAB Functions from .NET”
- “编写适用于 MATLAB 的 COM 应用程序”

20

将 Web 服务与 MATLAB 结合使用

Python 接口主题

从 MATLAB 访问 Python 模块 - 快速入门

您可以从 MATLAB 访问所有标准 Python 库内容。同样，您可以在第三方或用户创建的模块中使用该功能。要直接从 MATLAB 调用 Python 功能，请将 **py.** 前缀添加到要调用的 Python 函数的名称。

- 要调用 Python 标准库中的内容，请在 Python 函数或类名称的前面添加 **py.**。

```
py.list({'This','is a','list'}) % Call built-in function list
```

- 要调用可用模块中的内容，请在后跟 Python 函数或类名称的 Python 模块名称前面添加 **py.**。

```
py.textwrap.wrap('This is a string') % Call wrap function in module textwrap
```

无需导入模块便可使用它们。但您可以将 Python 名称导入到您的 MATLAB 函数中，方法与导入 MATLAB 包中的内容的方法相同。有关详细信息，请参阅了解 Python 和 MATLAB 导入命令 （第 21-34 页）。

MATLAB 还提供了一种直接从 MATLAB 在 Python 解释器中运行 Python 代码的方法。有关详细信息，请参阅“直接从 MATLAB 调用 Python 功能” （第 21-38 页）。

学习目标

本教程解释如何：

- 检查您计算机上的 Python 版本。
- 创建一个 Python 对象并对其调用方法。
- 显示 Python 模块的帮助。
- 创建专用的 Python **list**、**tuple** 和 **dict**（字典）类型
- 对 Python 对象调用与 MATLAB 函数同名的方法。
- 从您自己的 Python 模块调用功能。
- 查找示例。

验证 Python 配置

要在 MATLAB 中使用 Python，您的计算机上必须安装受支持的 Python 版本。要验证您是否安装了受支持的版本，请键入：

```
pyenv
```

```
ans =

  PythonEnvironment with properties:

         Version: "3.8"
      Executable: "C:\Users\aname\AppData\Local\Programs\Python\Python38\pythonw.exe"
         Library: "C:\Users\aname\AppData\Local\Programs\Python\Python38\python38.dll"
            Home: "C:\Users\aname\AppData\Local\Programs\Python\Python38"
          Status: NotLoaded
    ExecutionMode: OutOfProcess
```

如果 **Version** 属性的值为空，则您没有可用的受支持版本。有关安装 Python 的详细信息，请参阅“配置您的系统使用 Python” （第 21-10 页）。

在 MATLAB 中访问 Python 标准库模块

MATLAB 与您计算机上的 Python 解释器交互，让您可以访问所有标准库内容。例如，创建一个 Python **list** 数据类型。

```
res = py.list({'Name1','Name2','Name3'})
```

```
res =

  Python list with no properties.

    ['Name1', 'Name2', 'Name3']
```

MATLAB 会识别 Python 对象，并自动将 MATLAB 元胞数组转换为适当的 Python 类型。

您可以对一个对象调用 Python 方法。要显示对 **list** 对象可用的方法，请键入 **methods(py.list)**。例如，使用 Python **append** 函数更新列表 **res**。

```
res.append('Name4')
res
```

```
res =

  Python list with no properties.

    ['Name1', 'Name2', 'Name3', 'Name4']
```

要将 **list** 变量转换为 MATLAB 变量，请对列表调用 **cell** 并对列表的元素调用 **char**。

```
mylist = cellfun(@char,cell(res),'UniformOutput',false)
```

```
mylist =

  1×4 cell array

    {'Name1'}    {'Name2'}    {'Name3'}    {'Name4'}
```

在 MATLAB 中显示 Python 文档

您可以在 MATLAB 中显示 Python 函数的帮助文本。例如：

```
py.help('list.append')
```

```
Help on method_descriptor in list:

list.append = append(...)
    L.append(object) -> None -- append object to end
```

键入 **py.** 时的 Tab 键自动填充不显示可用的 Python 功能。有关详细信息，请参阅 “Python 函数的帮助” （第 21-34 页）。

创建列表、元组和字典类型

下表显示用于创建 **list**、**tuple** 和 **dict** 类型的语句。左边是从 Python 解释器运行的语句。右边是 MATLAB 语句。

Python list - []	MATLAB py.list
>>> ['Robert', 'Mary', 'Joseph']	>> py.list({'Robert','Mary','Joseph'})
>>> [[1,2],[3,4]]	>> py.list({py.list([1,2]),py.list([3,4])})

Python tuple - ()	MATLAB py.tuple
>>> ('Robert', 19, 'Biology')	>> py.tuple({'Robert',19,'Biology'})

Python dict - {}	MATLAB py.dict
>>> {'Robert': 357, 'Joe': 391, 'Mary': 229}	>> py.dict(pyargs(... 'Robert',357,'Mary',229,'Joe',391)) 有关传递关键字参数的信息，请参阅 pyargs。

方法和函数的优先顺序

如果 Python 类为 Python 类型定义了与 MATLAB 转换器方法同名的方法，则 MATLAB 会调用 Python 方法。这意味着您无法对该类的对象调用 MATLAB 转换器方法。

例如，如果 Python 类定义了 **char** 方法，以下语句将调用 Python 方法。

```
char(obj)
```

要使用 MATLAB **char** 函数，请键入：

```
char(py.str(obj))
```

访问其他 Python 模块

您可以在 MATLAB 中使用自己的 Python 代码和第三方模块。内容必须位于 Python 路径中。安装第三方模块会将内容放在 Python 路径中。如果您创建自己的模块，则由您负责将它们放到路径中。

有关示例，请参阅“调用用户定义的 Python 模块” （第 21-7 页）。

Python 示例

有关可在 MATLAB 实时编辑器中打开的示例代码，请在 “从 MATLAB 中调用 Python”页上查找“精选示例”。有关搜索 MATLAB 示例的信息，请参阅 “MATLAB 代码示例”。

有关使用在线数据集的示例，请参阅此 MathWorks 博客文章。

另请参阅

pyenv

详细信息

- “在 MATLAB 中调用 Python 函数以使段落文本换行” （第 21-5 页）
- “配置您的系统使用 Python” （第 21-10 页）
- “直接从 MATLAB 调用 Python 功能” （第 21-38 页）

在 MATLAB 中调用 Python 函数以使段落文本换行

以下示例说明如何在 MATLAB® 中使用 Python® 语言函数和模块。该示例从 Python 标准库中调用一个文本格式的模块。

MATLAB 支持 Python 的参考实现，通常称为 CPython。如果您在 Mac 或 Linux 平台上，会默认安装 Python。如果您在 Windows 上，则需要安装一个分发版，这种版本您可以在 https://www.python.org/downloads/ 找到。有关详细信息，请参阅“安装支持的 Python 实现” （第 21-10 页）。

使用 Python `textwrap` 模块

MATLAB 具有 Python 标准库的大量等效功能，但并非全部。例如，`textwrap` 是一个模块，它可使用回车和其他便捷方式格式化文本块。MATLAB 同样提供了一个 `textwrap` 函数，但该函数允许文本为适应 UI 控件而换行。

创建一个文本段落来进行测试。

```
T = "We at MathWorks believe in the importance of engineers and scientists. They increase human knowledge ar
```

将 Python 字符串转换为 MATLAB 字符串

通过在 `textwrap.wrap` 函数名称之前输入字符 `py.` 来调用该函数。请勿输入 `import textwrap`。

```
W = py.textwrap.wrap(T);
whos W
```

```
  Name      Size            Bytes  Class     Attributes

  W         1x3                 8  py.list
```

`W` 是一个 Python 列表，MATLAB 将其显示为 `py.list` 类型。每个元素均为一个 Python 字符串。

```
W{1}
```

```
ans =
  Python str with no properties.

    We at MathWorks believe in the importance of engineers and scientists.
```

将 `py.list` 转换为字符串数组。

```
wrapped = string(W);
whos wrapped
```

```
  Name        Size            Bytes  Class     Attributes

  wrapped     1x3               514  string
```

```
wrapped{1}
```

```
ans =
'We at MathWorks believe in the importance of engineers and scientists.'
```

自定义段落

使用关键字参数自定义段落的输出。

前面的代码使用 **wrap** 便利函数，但模块使用 **py.textwap.TextWrapper** 功能提供更多选项。若要使用这些选项，请使用 https://docs.python.org/2/library/textwrap.html#textwrap.TextWrapper 所述的关键字参数调用 **py.textwap.TextWrapper**。

使用 **width** 关键字将文本格式化为宽度为 30 个字符。**initial_indent** 和 **subsequent_indent** 关键字使每一行以 MATLAB 所用的注释字符 % 开头。

```
tw = py.textwrap.TextWrapper(initial_indent="% ",subsequent_indent="% ",width=int32(30));
W = wrap(tw,T);
```

转换为 MATLAB 参数并显示结果。

```
message = string(W);
fprintf("%s\n", message{:})
```

```
% We at MathWorks believe in
% the importance of engineers
% and scientists. They
% increase human knowledge and
% profoundly improve our
% standard of living.
```

了解更多信息

记得 Python 也可能提供 MATLAB 用户所需的库就足够了。如果您想了解 MATLAB 和 Python 之间的数据移动情况，包括元组和字典等 Python 数据类型，请参阅“从 MATLAB 中调用 Python”。

调用用户定义的 Python 模块

此示例说明如何从以下 Python 模块调用方法。本文档中的示例会使用此模块。此示例说明如何在 MATLAB 中创建该模块。如果您在 Python 编辑器中创建 **mymod.py**，请确保该模块位于 Python 搜索路径上。此示例还可指导经验不丰富的 Python 用户如何获得调用该函数的帮助。

- 将当前文件夹更改为可写文件夹。
- 在 MATLAB 编辑器中打开一个新文件。
- 复制下列命令并将该文件另存为 **mymod.py**。

```
# mymod.py
"""Python module demonstrates passing MATLAB types to Python functions"""
def search(words):
    """Return list of words containing 'son'"""
    newlist = [w for w in words if 'son' in w]
    return newlist

def theend(words):
    """Append 'The End' to list of words"""
    words.append('The End')
    return words
```

- 从 MATLAB 命令提示符下，将当前文件夹添加到 Python 搜索路径。

```
if count(py.sys.path,'') == 0
    insert(py.sys.path,int32(0),'');
end
```

- 要了解如何调用函数，请阅读 **mymod.py** 源文件中 **search** 函数的函数签名。该函数接受一个输入参数 **words**。

```
def search(words):
```

- 阅读 **mymod.py** 源文件中的函数帮助。根据 Python 网站文档，帮助位于“作为模块、函数、类或方法定义中的第一条语句出现的字符串文字”中。**search** 的帮助是：

```
"""Return list of words containing 'son'"""
```

该函数返回一个列表。

- 在 MATLAB 中创建一个输入参数，即名称列表。

```
N = py.list({'Jones','Johnson','James'})
```

```
N =

  Python list with no properties.

    ['Jones', 'Johnson', 'James']
```

- 调用 **search** 函数。在模块名称和函数名称前键入 **py.**。

```
names = py.mymod.search(N)
```

```
names =

  Python list with no properties.

    ['Johnson']
```

该函数返回 **py.list** 值。

- 原始输入 **N** 未更改。

```
N
```

```
N =

  Python list with no properties.

    ['Jones', 'Johnson', 'James']
```

重新加载经过修改的用户定义的 Python 模块

此示例说明如何在进程内运行 Python 解释器的同时重新加载修改后的 Python 模块。有关替代方法，请参阅“重新加载进程外 Python 解释器” （第 21-20 页）。

创建 Python 模块

将当前文件夹更改为可写文件夹。在 MATLAB 编辑器中打开一个新文件。

复制下列用于定义 **myfunc** 函数的语句并将文件另存为 **newmod.py**。

```
# newmod.py
def myfunc():
    """Display message."""
    return 'version 1'
```

调用 **myfunc**。

```
py.newmod.myfunc
```

```
ans =

  Python str with no properties.

    version 1
```

修改模块

修改该函数，用以下代码替换 **return** 语句：

```
    return 'version 2'
```

保存文件。

卸载模块

```
clear classes
```

MATLAB 删除工作区中的所有变量、脚本和类。

导入经过修改的模块

```
mod = py.importlib.import_module('newmod');
```

在 Python 中重新加载模块

```
py.importlib.reload(mod);
```

调用更新后的模块中的函数

调用更新后的 **myfunc** 函数。

```
py.newmod.myfunc
```

```
ans =

  Python str with no properties.

    version 2
```

另请参阅

pyenv

详细信息

- “配置您的系统使用 Python” （第 21-10 页）
- “重新加载进程外 Python 解释器” （第 21-20 页）

配置您的系统使用 Python

Python 支持

要在 MATLAB 中调用 Python 模块，您必须在您的系统上安装受支持的参考实现 (CPython) 版本。安装一个发行版，例如 https://www.python.org/downloads/ 上提供的发行版。MATLAB 不支持从 Microsoft 商店安装的 CPython 版本。有关支持的版本信息，请参阅 MATLAB 产品（按版本）兼容的 Python 版本。如果您在 Linux 或 Mac 平台上，则已默认安装 Python。如果您使用的是 Windows 平台，则需要安装一个发行版（如果尚未安装）。有关详细信息，请参阅“安装支持的 Python 实现” （第 21-10 页）。

要验证您的系统上是否安装了 Python，请从系统提示符下打开 Python 解释器，并调用 Python 函数。

默认情况下，MATLAB 根据您的系统路径选择 Python 的版本。要在 MATLAB 中查看系统路径，请使用 **getenv('path')** 命令。要确定使用的是哪个版本的 MATLAB，请调用 **pyenv** 函数。

```
pe = pyenv;
pe.Version
```

```
ans =

  "3.8"
```

由 **pyenv** 设置的值可跨 MATLAB 会话而保持不变。如果您有多个受支持的版本，请使用 **pyenv** 显示 MATLAB 当前使用的版本。MATLAB 会在您键入 Python 语句时自动选择并加载一个 Python 版本。例如，要调用 **funcname**，请键入：

```
py.funcname
```

要更改版本，请执行以下操作：

- 如果在单一 MATLAB 会话中的 **InProcess ExecutionMode** 中加载了 Python，则重新启动 MATLAB 并使用新版本信息运行 **pyenv**。
- 如果在 **OutOfProcess** 模式下加载 Python，则调用 **terminate** 并使用新版本信息运行 **pyenv**。

安装支持的 Python 实现

- 访问 https://www.python.org/downloads/ 并滚动到 **Looking for a specific release**（查找特定版本）部分。
- 找到您需要的版本，然后点击 **Download**（下载）。有关支持的版本信息，请参阅 MATLAB 产品（按版本）兼容的 Python 版本。
- 点击 64 位版本所需的格式，然后按照在线说明进行操作。

> **注意** 要在 Microsoft Windows 系统上安装 64 位 MATLAB 的 2.7 版本，请选择 64 位 Python 版本，称为 Windows x86-64 MSI installer。

如果您收到错误消息 “无法解析名称 py.myfunc” （第 21-18 页），则可能存在安装问题。

在 Windows 平台上设置 Python 版本

在 Windows 平台上，使用以下任一方式：

pyenv('Version','version')

或

pyenv('Version','executable')

其中 **executable** 是 Python 可执行文件的完整路径。

注意 如果您下载了一个 Python 解释器，但未将其注册到 Windows 注册表中，请使用：

pyenv('Version','executable')

在 Windows 平台上下载 Python 的 64 位版本

Python 的架构必须与 MATLAB 的架构匹配。有关详细信息，请参阅“安装支持的 Python 实现” （第 21-10 页）。

在 Mac 和 Linux 平台上设置 Python 版本

要设置版本，请键入：

pyenv('Version','executable')

其中 **executable** 是 Python 可执行文件的完整路径。

编译 Python 可执行文件的要求

在 Linux 和 Mac 系统上，如果您要编译 Python 可执行文件，请使用 **--enable-shared** 选项对编译进行配置。

另请参阅

pyenv

详细信息

- MATLAB 产品（按版本）兼容的 Python 版本
- “无法解析名称 py.myfunc” （第 21-18 页）

外部网站

- https://www.python.org/downloads/

MATLAB 到 Python 的数据类型映射

当调用 Python 函数时，MATLAB 将 MATLAB 数据转换为最适合在 Python 语言中表达该数据的类型。

将标量值传递给 Python

MATLAB 输入参数类型 - 仅标量值	生成的 Python py. 类型	示例
double（实数） single（实数）	float	“在 MATLAB 中使用 Python 数值变量” （第 21-21 页）
double（复数） single（复数）	complex	z = complex(1,2); py.cmath.polar(z) ans = Python tuple with no properties. (2.23606797749979, 1.1071487177940904)
int8 uint8 int16 uint16 int32	int	
uint32 int64 uint64	int long（仅限 2.7 版本）	
NaN	float("nan")	
Inf	float("inf")	
string 标量 char 向量	str	“在 MATLAB 中使用 Python str 变量” （第 21-24 页）
string 中的 <missing> 值	None	py.list({string(missing),'Value'}) ans = Python list with no properties. [None, 'Value']
logical	bool	
结构体	dict	“在 MATLAB 中使用 Python dict 变量” （第 21-32 页）
Python 对象 - py.type	type	
函数句柄 @py.module.function（仅限于指向 Python 函数的函数句柄）	module.function	“Pass Python Function to Python map Function”

将向量传递给 Python

MATLAB 输入参数类型 - 1 ×N 向量	生成的 Python 类型
double（实数）	**array.array('d')**
single（实数）	**array.array('f')**
int8（实数）	**array.array('b')**
uint8（实数）	**array.array('B')**
int16（实数）	**array.array('h')**
uint16（实数）	**array.array('H')**
int32（实数）	**array.array('i')**
uint32（实数）	**array.array('I')**
int64（实数）- 不支持 Windows 上的 Python 2.7	**array.array('q')**
uint64（实数）- 不支持 Windows 上的 Python 2.7	**array.array('Q')**
double（复数） **single**（复数） **int8**（复数） **uint8**（复数） **int16**（复数） **uint16**（复数） **int32**（复数） **uint32**（复数）	**memoryview**
logical	**memoryview**
char 向量 **string** 标量	**str**
包含大于 127 的值的 **char** 数组（仅限版本 2.7）	**unicode**
cell 向量	**tuple**

将矩阵和多维数组传递给 Python

Python 语言提供访问内存缓冲区（如存储在 MATLAB 数组中的数据）的协议。MATLAB 为 MATLAB 数组实现此 Python 缓冲区协议，以便直接从 Python 代码中读取 MATLAB 数组，从而在与 MATLAB 相同的进程中运行，而无需复制数据。

许多 Python 函数直接从 Python 使用 MATLAB 数组，而不将其转换为原生 Python 类型。某些函数可能需要特定类型，例如 **numpy.ndarray**，或可能会修改数组中的数据。这些函数可能接受 MATLAB 数组并将数据复制到所需的类型中。其他函数则不然，如果您未传递所需的类型，可能就会显示错误。要将数据传递给这些函数，请首先从 MATLAB 数据创建所需的 Python 类型，然后将其传递给 Python 函数。例如，要创建数组 **p** 以传递给需要 **numpy.array** 类型数据的 Python 函数，请键入：

```
p = py.numpy.array(magic(3))
```

```
p =

  Python ndarray:
```

```
     8     1     6
     3     5     7
     4     9     2

  Use details function to view the properties of the Python object.

  Use double function to convert to a MATLAB array.
```

在 Python 中不支持 MATLAB 稀疏数组。请参阅“不支持的 MATLAB 类型” （第 21-17 页）。

参数错误故障排除

如果 Python 函数需要特定的 Python 多维数组类型，例如 **numpy.ndarray**，MATLAB 将显示消息，并提示应该如何继续。如果问题产生的原因有可能是将矩阵或多维数组作为参数传递，请执行以下操作。

1 查阅 Python 函数的文档，找出参数需要的类型。

2 在 MATLAB 中创建该类型的 Python 对象，并将其传递给 Python 函数。

例如，假设以下代码返回错误。

```
a = [1 2; 3 4];
py.pyfunc(a)
```

如果 **pyfunc** 的文档指定需要的类型为 **numpy.ndarray**，请尝试进行以下转换：

```
py.pyfunc(numpy.ndarray(a))
```

如果错误仍然存在，请通过查看 Python 异常中的其他信息来确定根本原因。

自动将 Python 类型转换为 MATLAB 类型

MATLAB 自动将从 Python 函数返回的下列数据类型转换为 MATLAB 类型。要转换其他类型，请参阅“将 Python 类型显式转换为 MATLAB 类型” （第 21-14 页）。

Python 返回类型，如 Python 中所显示	生成的 MATLAB 类型 - 标量
float	**double**
complex	复数 **double**
int（仅限版本 2.7）。 对于 Python 版本 3.x **int**，您必须显式转换。请参阅 “将 Python 类型显式转换为 MATLAB 类型”（第 21-14 页）。	**int64**
bool	**logical**
所有其他 Python 类型 - **type**	Python 对象 - **py.type**

将 Python 类型显式转换为 MATLAB 类型

如果 Python 函数的输出实现 Python 缓冲区协议（如 **numpy.ndarray**）并且是数值或逻辑值，则 MATLAB 显示：

- 实际的 Python 类型

- 底层数据
- 对应的 MATLAB 转换函数。使用此函数可将 Python 对象完全转换为 MATLAB 数组。

使用下列 MATLAB 函数将 Python 数据类型转换为 MATLAB 类型。

Python 返回类型或协议，如在 MATLAB 中所显示	MATLAB 转换函数	示例
py.str（版本 3.x）	**string** **char**	“在 MATLAB 中使用 Python str 变量” （第 21-24 页）
py.str（版本 2.7）	**string** **char** **uint8**	
py.unicode	**string** **char**	
使用 **__str__** 方法的对象	**char**	**py.help('datetime.date.__str__')** **Help on wrapper_descriptor in datetime.date** **datetime.date.__str__ = __str__(self, /)** **Return str(self).** **d = py.datetime.date(...** **int32(2020),int32(3),int32(4));** **char(d)** **ans = '2020-3-04'**
py.int **py.long** **py.float** **py.bool**	数值函数： **double** **single** **int8** **uint8** **int16** **uint16** **int32** **uint32** **int64** **uint64**	
	logical	
py.bytes	**uint8**	
py.array.array **py.numpy.ndarray** **py.memoryview** 您可以将任何格式的 **py.array.array** 和 **py.memoryview** 对象转换为所需的 MATLAB 类型。	数值函数： **double** **single** **int8** **uint8** **int16** **uint16** **int32** **uint32** **int64** **uint64**	“在 MATLAB 中使用 Python 数值变量” （第 21-21 页），例如，在 MATLAB 中使用 Python 整数数组类型。

Python 返回类型或协议，如在 MATLAB 中所显示	MATLAB 转换函数	示例
py.list 和 **py.tuple**	**double** **single** **int8** **uint8** **int16** **uint16** **int32** **uint32** **int64** **uint64** **logical** **string** **cell**	“在 MATLAB 中使用 Python list 变量” (第 21-26 页) “在 MATLAB 中使用 Python tuple 变量” (第 21-30 页) 有关详细信息，请参阅“Error Converting Elements of list or tuple”。
映射协议；例如，**py.dict**	**struct**	“在 MATLAB 中使用 Python dict 变量” (第 21-32 页)

例如，Python 函数返回此数组 **p**：

```
p =

  Python ndarray:

    8    1    6
    3    5    7
    4    9    2

    Use details function to view the properties of the Python object.

    Use double function to convert to a MATLAB array.
```

您可以通过键入以下内容将其转换为 MATLAB 矩阵 **P**：

```
P = double(p)
```

```
P = 3×3
    8    1    6
    3    5    7
    4    9    2
```

如果需要 **p** 的 Python 属性的特定信息，请键入：

```
details(p)
```

```
    py.numpy.ndarray handle with properties:

        T: [1×1 py.numpy.ndarray]
     base: [1×1 py.NoneType]
   ctypes: [1×1 py.numpy.core._internal._ctypes]
     data: [1×3 py.memoryview]
    dtype: [1×1 py.numpy.dtype[float64]]
    flags: [1×1 py.numpy.flagsobj]
     flat: [1×1 py.numpy.flatiter]
     imag: [1×1 py.numpy.ndarray]
 itemsize: [1×1 py.int]
```

```
    nbytes: [1×1 py.int]
      ndim: [1×1 py.int]
      real: [1×1 py.numpy.ndarray]
     shape: [1×2 py.tuple]
      size: [1×1 py.int]
   strides: [1×2 py.tuple]

  Methods, Events, Superclasses
```

如果 Python 模块在其 **__doc__** 特性中提供内容，MATLAB 将链接到该信息。

不要使用 Python 对象作为字典的键

您无法将 Python 对象作为键参数传递给 MATLAB **dictionary** 函数，也无法将其作为输入传递给 **keyMatch** 函数。

不支持的 MATLAB 类型

Python 不支持下列 MATLAB 类型。

- 多维 **char** 或 **cell** 数组
- 结构体数组
- 稀疏数组
- **categorical**、**table**、**containers.Map**、**datetime** 类型
- MATLAB 对象
- **meta.class** (**py.class**)

另请参阅

相关示例

- “在 MATLAB 中使用 Python 数值变量” (第 21-21 页)
- “在 MATLAB 中使用 Python str 变量” (第 21-24 页)
- “在 MATLAB 中使用 Python list 变量” (第 21-26 页)
- “在 MATLAB 中使用 Python tuple 变量” (第 21-30 页)
- “在 MATLAB 中使用 Python dict 变量” (第 21-32 页)

无法解析名称 py.myfunc

当您在 MATLAB 命令提示符下键入 **py.** 后接 Python 语句时，MATLAB 会自动加载 Python。如果 MATLAB 显示此消息，则对 **myfunc** 的调用发生了失败。

```
Unable to resolve the name py.myfunc
```

此页面有助于对加载失败进行故障排除。

Python 未安装

您的计算机上未安装受支持的 Python 版本。请参阅您的 MATLAB 版本的“配置您的系统使用 Python”（第 21-10 页），然后从 **https://www.python.org/downloads/** 下载并安装 Python。

在 Linux 和 Mac 系统上，如果从源文件编译 Python，请使用 **--enable-shared** 选项对该编译进行配置。

要验证系统上是否安装了 Python，请检查 **PythonEnvironment Version** 属性。

```
pe = pyenv;
if isempty(pe.Version)
   disp "Python not installed"
end
```

Windows 平台上的 Python 的 64 位/32 位版本

您为 64 位版本的 MATLAB 安装了 32 位版本的 Python。您必须安装 Python 的 64 位版本。

MATLAB 找不到 Python

Python 位于非标准位置。要提供 Python 可执行文件的路径，请使用 **pyenv** 函数。例如：

```
pyenv(Version="C:\Users\uname\WinPython-64bit-3..2.1\python-3..2.amd64\python.exe")
```

在 Windows 系统上，在 Windows 注册表中找不到 Python。如果您下载了 Python 解释器，但没有在 Windows 注册表中注册，请指定 Python 位置：

```
pyenv(Version="executable")
```

在用户定义的 Python 模块中的错误

用户定义的 Python 模块发生错误。要测试您的模块 **mymod** 是否包含错误，请键入：

```
py.importlib.import_module('mymod')
```

如果 Python 检测到该模块中有错误，则 MATLAB 会显示一条 Python 错误消息。

您也可以在 Python 命令提示符下执行等效语句，以获取该 Python 错误消息。

改正错误后，要访问更新后的模块，请重新启动 MATLAB，并将其添加到搜索路径中。

Python 模块不在 Python 搜索路径上

如果 **command** 是有效的 Python 命令，请确保 Python 模块在 Python 搜索路径上。要测试模块 **mymod** 是否在搜索路径上，请键入：

```
py.importlib.import_module('mymod')
```

如果 Python 找不到该模块，MATLAB 将显示一条 Python 错误消息。

要将文件夹 **modpath** 中的 **mymod** 添加到路径，请键入：

```
P = py.sys.path;
if count(P,'modpath') == 0
   insert(P,int32(0),'modpath');
end
```

Python 搜索路径与 MATLAB 当前会话中加载的 Python 解释器相关联。您可以在 MATLAB 中修改搜索路径，但如果您在 MATLAB 之外运行解释器的其他实例，则修改将不会保留。

模块名称冲突

如果您调用的 Python 模块与标准库中的模块或系统上安装的任何第三方模块同名，则 MATLAB 可能会加载错误的模块。

Python 尝试在错误的模块中执行 myfunc

如果 **myfunc** 在用户定义的模块中，请确保该模块不与 Python 标准库中的模块或系统上的任何第三方模块发生名称冲突。

另请参阅

pyenv

详细信息

- “配置您的系统使用 Python” （第 21-10 页）

外部网站

- https://www.python.org/downloads/

重新加载进程外 Python 解释器

当您在进程外运行 Python 解释器时，您可以终止解释器并启动新解释器（可能使用不同版本设置），而无需重新启动 MATLAB。

要重新加载进程内 Python 解释器，请参阅示例“重新加载经过修改的用户定义的 Python 模块” （第 21-8 页）。

此示例假设您有 Python 3.8 和 3.10 版本。如果您已加载进程内解释器，请重新启动 MATLAB。

```
pe = pyenv;
if pe.Status == 'NotLoaded'
    pyenv(ExecutionMode="OutOfProcess",Version="3.8");
end
py.list; % Call a Python function to load interpreter
pyenv
```

```
ans =
  PythonEnvironment with properties:

         Version: "3.8"
      Executable: "C:\Python38\pythonw.exe"
         Library: "C:\WINDOWS\system32\python38.dll"
            Home: "C:\Python38"
          Status: Loaded
    ExecutionMode: OutOfProcess
       ProcessID: "15176"
     ProcessName: "MATLABPyHost"
```

重新加载 Python 3.10 版本解释器。

```
terminate(pyenv)
pyenv(Version="3.10");
py.list; % Reload interpreter
pyenv
```

```
ans =
  PythonEnvironment with properties:

         Version: "3.10"
      Executable: "C:\Python310\pythonw.exe"
         Library: "C:\WINDOWS\system32\python310.dll"
            Home: "C:\Python310"
          Status: Loaded
    ExecutionMode: OutOfProcess
       ProcessID: "24840"
     ProcessName: "MATLABPyHost"
```

另请参阅

详细信息

- “重新加载经过修改的用户定义的 Python 模块” （第 21-8 页）

在 MATLAB 中使用 Python 数值变量

此示例说明如何在 MATLAB® 中使用 Python® 数值类型。

在 MATLAB 中使用 Python 数值类型

当调用接受数值输入参数的 Python 函数时，MATLAB 会将双精度值转换为最适合在 Python 语言中表示该数据的类型。例如，要调用 Python **math** 模块中的三角函数，请传递 MATLAB 双精度值。

```
pynum = py.math.radians(90)
```

```
pynum = 1.5708
```

对于返回 Python **float** 类型的函数，MATLAB 会自动将该类型转换为双精度类型。

```
class(pynum)
```

```
ans =
'double'
```

对于返回整数类型的 Python 函数，MATLAB 会自动将该类型转换为 **int64**。例如，**bit_length** 函数返回将二进制整数表示为 **int** 值所需的位数。

```
py.int(intmax).bit_length
```

```
ans = int64
   31
```

用数值 iterable 参数调用 Python 方法

Python **math.fsum** 函数对 **iterable** 输入参数中的浮点值求和。您可以将 MATLAB 向量传递给此函数。例如，打开 MATLAB **patients.mat** 数据文件并读取数值数组 **Height**。

```
load patients.mat
class(Height)
```

```
ans =
'double'
```

```
size(Height)
```

```
ans = 1×2

   100     1
```

当您将此参数传递给 Python 时，MATLAB 自动将数值转换为 Python 数值且 Python 会对向量值进行迭代。

```
py.math.fsum(Height)
```

```
ans = 6707
```

在 MATLAB 中使用 Python array 类型

假设您有一个 Python 函数，它返回以下双精度类型的 Python **array.array**。

```
P = py.array.array('d', 1:5)
```

```
P =
  Python array with properties:

    itemsize: 8
    typecode: [1×1 py.str]

    array('d', [1.0, 2.0, 3.0, 4.0, 5.0])
```

要将 **P** 传递给 MATLAB 函数 **sum**，请将 **P** 转换为双精度类型的 MATLAB 数组。

```
sum(double(P))
```

```
ans = 15
```

在 MATLAB 中使用 Python 整数 array 类型

假设您有以下 Python 数组。对该数组调用 Python **reverse** 函数，然后将结果转换为 MATLAB 数组。

```
arr = py.array.array('i',[int32(5),int32(1),int32(-5)])
```

```
arr =
  Python array with properties:

    itemsize: 4
    typecode: [1×1 py.str]

    array('i', [5, 1, -5])
```

```
arr.reverse
A = int32(arr)
```

```
A = 1×3 int32 row vector

  -5    1    5
```

默认数值类型

默认情况下，MATLAB 中的数值是 **double** 类型。默认情况下，Python 中的数值（没有小数部分）是整数类型。这种差异会导致在将数值传递给 Python 函数时出现混淆。

例如，当您将下列 MATLAB 数值传递给 Python **datetime** 函数时，Python 会将它们读取为 **float** 类型并显示错误：

```
d = py.datetime.date(2014,12,31)
```

```
Python Error: TypeError: integer argument expected, got float
```

要更正该错误，请将每个数值显式转换为整数类型：

```
d = py.datetime.date(int32(2014),int32(12),int32(31))
```

```
d =
  Python date with properties:

     day: 31
   month: 12
```

```
    year: 2014

   2014-12-31
```

为什么我在显示数值时会看到属性?

MATLAB 将所有 Python 类型显示为对象，其中包括对象属性列表。对于数值类型，MATLAB 在最后一行显示预期的输出值。

```
py.int(5)

ans =
  Python int with properties:

    denominator: 1
           imag: 0
      numerator: 5
           real: 5

    5
```

在 MATLAB 中使用 Python str 变量

此示例说明如何在 MATLAB® 中使用 Python® **str** 变量。

用 str 输入参数调用 Python 函数

要调用接受 **str** 输入参数的 Python 函数，请传递 MATLAB 字符串或字符向量。MATLAB 自动将这些值转换为 Python **str** 类型。

例如，Python **os.listdir** 函数获取有关文件夹内容的信息，指定为类型 **str**。创建一个表示有效文件夹的字符向量并调用 **os.listdir**。示例文件夹的数量基于您安装的产品。

```
folder = fullfile(matlabroot,'help','examples');
F = py.os.listdir(folder);
exFolders = py.len(F)
```

```
exFolders =
  Python int with properties:

    denominator: [1×1 py.int]
           imag: [1×1 py.int]
      numerator: [1×1 py.int]
           real: [1×1 py.int]

    267
```

在 MATLAB 中使用 Python str 类型

在 MATLAB 中，Python 字符串是 **py.str** 变量。要在 MATLAB 中使用此变量，请调用 **char**。例如，Python **os.path.pathsep** 函数返回 Python 路径分隔符，即分号 (;)。

```
p = py.os.path.pathsep
```

```
p =
  Python str with no properties.

    ;
```

要在路径名称之间插入此字符，请键入：

```
['mypath' char(p) 'nextpath']
```

```
ans =
'mypath;nextpath'
```

读取 Python 字符串中的元素

您可以像对 MATLAB 字符串进行索引一样对 Python 字符串进行索引。创建一个 MATLAB 字符向量并显示某字符范围。

```
str = 'myfile';
str(2:end)
```

```
ans =
'yfile'
```

将该字符向量转换为 Python **str** 类型并显示相同的字符。

```
pstr = py.str(str);
pstr(2:end)
```

```
ans =
  Python str with no properties.

    yfile
```

传递 MATLAB 反斜杠控制字符

要将反斜杠控制字符 (\) 作为 Python **str** 类型传递，请通过调用 MATLAB **sprintf** 函数插入新行控制字符 **\n**。Python 会用一个新行替换 **\n**。

```
py.str(sprintf('The rain\nin Spain.'))
```

```
ans =
  Python str with no properties.

    The rain
    in Spain.
```

如果没有 **sprintf** 函数，MATLAB 和 Python 都会将 \ 解释为字面值反斜杠。

```
py.str('The rain\nin Spain.')
```

```
ans =
  Python str with no properties.

    The rain\nin Spain.
```

将此字符串传递给 Python 字符串方法 **split**。Python 将 MATLAB 字符向量视为原始字符串，并添加 \ 字符来保留原始反斜杠。

```
split(py.str('The rain\nin Spain.'))
```

```
ans =
  Python list with no properties.

    ['The', 'rain\\nin', 'Spain.']
```

在 MATLAB 中使用 Python list 变量

此示例说明如何在 MATLAB® 中使用 Python® **list** 变量。

要调用接受 **list** 输入参数的 Python 函数，请创建一个 **py.list** 变量。要将列表转换为 MATLAB 变量，请调用 **cell** 函数，然后为列表中的每个元素调用适当的转换函数。

调用接受 list 输入参数的 Python 函数

Python **len** 函数返回容器中的项目数，其中包括一个 **list** 对象。

```
py.help('len')
```

```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

调用 **os.listdir** 以创建一个名为 **P** 的由程序组成的 Python **list**。

```
P = py.os.listdir("C:\Program Files\MATLAB");
class(P)
```

```
ans =
'py.list'
```

显示程序的数量。

```
py.len(P)
```

```
ans =
  Python int with properties:

    denominator: [1×1 py.int]
           imag: [1×1 py.int]
      numerator: [1×1 py.int]
           real: [1×1 py.int]

    9
```

显示一个元素。

```
P{2}
```

```
ans =
  Python str with no properties.

    R2016b
```

对 Python 列表进行索引

使用 MATLAB 索引显示列表中的元素。例如，显示 **list** 中的最后一个元素。MATLAB 返回一个 Python **list**。

```
P(end)
```

```
ans =
  Python list with no properties.

    ['R2021a']
```

您也可以在 **for** 循环中循环访问该列表。

```
for n = P
   disp(n{1})
end
```

```
  Python str with no properties.

    R2014b

  Python str with no properties.

    R2016b

  Python str with no properties.

    R2017b

  Python str with no properties.

    R2018b

  Python str with no properties.

    R2019a

  Python str with no properties.

    R2019b

  Python str with no properties.

    R2020a

  Python str with no properties.

    R2020b

  Python str with no properties.

    R2021a
```

将 Python list 类型转换为 MATLAB 类型

以下代码使用 MATLAB 变量显示 **list P** 中的名称。调用 **cell** 以转换该列表。该列表由 Python 字符串组成，因此调用 **char** 函数来转换元胞数组的元素。

```
cP = cell(P);
```

每个元胞元素名称均为一个 Python 字符串。

```
class(cP{1})
```

```
ans =
'py.str'
```

将 Python 字符串转换为 MATLAB 数据。

```
mlP = string(cell(P));
```

显示名称。

```
for n = 1:numel(cP)
   disp(mlP{n})
end
```

```
R2014b
R2016b
R2017b
R2018b
R2019a
R2019b
R2020a
R2020b
R2021a
```

在 MATLAB 中使用 Python 数值类型列表

Python **list** 包含任何类型的元素，并且可以包含混合类型的元素。以下代码中使用的 MATLAB **double** 函数假设 Python **list** 的所有元素均为数值。

假设您有一个 Python 函数，它返回名为 **P** 的由整数组成的 **list**。要运行此代码，请用以下值创建该变量。

```
P = py.list({int32(1), int32(2), int32(3), int32(4)})
```

```
P =
  Python list with no properties.

    [1, 2, 3, 4]
```

显示值的数值类型。

```
class(P{1})
```

```
ans =
'py.int'
```

将 **P** 转换为 MATLAB 元胞数组。

```
cP = cell(P);
```

将元胞数组转换为 **double** 类型的 MATLAB 数组。

```
A = cellfun(@double,cP)
```

```
A = 1×4

     1     2     3     4
```

读取嵌套 list 类型的元素

以下代码访问包含 list 元素的 Python list 变量的元素。假设您有此 list。

```
matrix = py.list({{1, 2, 3, 4},{'hello','world'},{9, 10}});
```

显示元素 'world'，它位于索引 (2,2) 处。

```
disp(char(matrix{2}{2}))
```

```
world
```

显示 Python 元素的步进范围

如果您使用切片来访问 Python 对象的元素，Python 中的格式是 start:stop:step。在 MATLAB 中，语法形式为 start:step:stop。

```
li = py.list({'a','bc',1,2,'def'});
li(1:2:end)
```

```
ans =
  Python list with no properties.

    ['a', 1.0, 'def']
```

在 MATLAB 中使用 Python tuple 变量

此示例说明如何在 MATLAB® 中使用 Python® **tuple** 变量。

将 tuple 转换为 MATLAB 变量

要将 **tuple** 转换为 MATLAB 元胞数组，请调用 **cell** 函数。

```
pStudent = py.tuple({'Robert',19,'Biology'})
```

```
pStudent =
  Python tuple with values:

    ('Robert', 19.0, 'Biology')

    Use string, double or cell function to convert to a MATLAB array.
```

```
S = cell(pStudent)
```

```
S=1×3 cell array
    {1×6 py.str}    {[19]}    {1×7 py.str}
```

读取 tuple 中的元素

使用 MATLAB 索引显示 **tuple** 中的元素。例如，显示 **pStudent** 的前两个元素。MATLAB 返回一个 **tuple** 变量。

```
pStudent(1:2)
```

```
ans =
  Python tuple with values:

    ('Robert', 19.0)

    Use string, double or cell function to convert to a MATLAB array.
```

显示一个元素。MATLAB 返回一个 Python 数据类型的元素。

```
pStudent{3}
```

```
ans =
  Python str with no properties.

    Biology
```

创建包含单个元素的 tuple

用单个元素创建一个 **tuple** 变量。MATLAB 对包含一个元素的 **tuple** 显示尾部逗号。

```
subject = py.tuple({'Biology'})
```

```
subject =
  Python tuple with values:
```

```
('Biology',)

Use string, double or cell function to convert to a MATLAB array.
```

在 MATLAB 中使用 Python dict 变量

此示例说明如何在 MATLAB® 中使用 Python® 字典 (**dict**) 变量。

要调用接受 **dict** 输入参数的 Python 函数，请创建一个 **py.dict** 变量。要将 **dict** 转换为 MATLAB 变量，请调用 **struct** 函数。

创建 Python dict 变量

创建一个 **dict** 变量以传递给 Python 函数。

```
studentID = py.dict(Robert=357,Mary=229,Jack=391)
```

```
studentID =
  Python dict with no properties.

    {'Robert': 357.0, 'Mary': 229.0, 'Jack': 391.0}
```

或者，创建一个 MATLAB 结构体并将其转换为 **dict** 变量。

```
S = struct("Robert",357,"Mary",229,"Jack",391);
studentID = py.dict(S)
```

```
studentID =
  Python dict with no properties.

    {'Robert': 357.0, 'Mary': 229.0, 'Jack': 391.0}
```

在 MATLAB 中使用 Python dict 类型

要将从 Python 函数返回的 **dict** 类型转换为 MATLAB 变量，请调用 **struct**。

假设您有一个 Python 函数，它在名为 **order** 的 **dict** 对象中返回菜单项和价格。要在 MATLAB 中运行以下代码，请创建此变量。

```
order = py.dict(soup=3.57,bread=2.29,bacon=3.91,salad=5.00)
```

```
order =
  Python dict with no properties.

    {'soup': 3.57, 'bread': 2.29, 'bacon': 3.91, 'salad': 5.0}
```

将 **order** 转换为 MATLAB 变量。

```
myOrder = struct(order)
```

```
myOrder = struct with fields:
    soup: 3.5700
   bread: 2.2900
   bacon: 3.9100
   salad: 5
```

使用 MATLAB 语法显示培根的价格。

```
price = myOrder.bacon
```

```
price = 3.9100
```

使用 Python 语法显示培根的价格。变量 **price** 为双精度类型，可在 MATLAB 中使用。

```
price = order{"bacon"}
```

```
price = 3.9100
```

字典有成对的键和值。使用 Python **keys** 函数显示变量 **order** 中的菜单项。

```
keys(order)
```

```
ans =
  Python dict_keys with no properties.

    dict_keys(['soup', 'bread', 'bacon', 'salad'])
```

使用 Python **values** 函数显示所有价格。

```
values(order)
```

```
ans =
  Python dict_values with no properties.

    dict_values([3.57, 2.29, 3.91, 5.0])
```

将 dict 参数传递给 Python 方法

Python **dict** 类有一个 **update** 方法。要运行以下代码，请创建包含患者和检测结果的 **dict** 变量。

```
patient = py.dict(name="John Doe", ...
test1= [], ...
test2= [220.0, 210.0, 205.0], ...
test3= [180.0, 178.0, 177.5]);
```

将患者姓名转换为 MATLAB 字符串。

```
string(patient{"name"})
```

```
ans =
"John Doe"
```

使用 **update** 方法更新并显示 **test1** 的结果。

```
update(patient,py.dict(test1=[79.0, 75.0, 73.0]))
P = struct(patient);
disp(["test1 results for "+string(patient{"name"})+": "+num2str(double(P.test1))])
```

```
test1 results for John Doe: 79  75  73
```

高级主题

了解 Python 和 MATLAB import 命令

import 语句在 MATLAB 中的功能与在 Python 中的功能不同。

在 MATLAB 中加载 Python 模块

Python 代码使用 **import** 语句来加载代码并使其可访问。当您在模块名称和函数名称前面键入 **py.** 时，MATLAB 会自动加载 Python。以下代码显示如何在 Python 模块 **textwrap** 中调用函数 **wrap**。

Python 代码	MATLAB 代码
import textwrap **pS1 = textwrap.wrap('This is a string')**	**S1 = py.textwrap.wrap('This is a string');**

小心 在 MATLAB 中，请勿键入：

import pythonmodule

切勿调用：

import py.*

如果执行了此调用，则 MATLAB 会调用 Python 函数，而不是同名的 MATLAB 函数。这可能会导致意外的行为。如果您键入以下 **import** 命令，则必须调用 MATLAB 命令：

clear import

缩短类或函数名称

借助 Python **from...import** 语句，您可以在不使用完全限定名称的情况下引用模块。在 MATLAB 中，使用 **import** 函数。以下代码显示如何在 Python 模块 **textwrap** 中引用函数 **wrap**。由于 **wrap** 不是 MATLAB 函数，您可以使用 **import** 函数缩短调用语法。调用此命令后，您不需要键入包 (**py**) 和模块 (**textwrap**) 名称。

Python 代码	MATLAB 代码
import textwrap **pS1 = textwrap.wrap('This is a string')** **from textwrap import wrap** **pS2 = wrap('another string')**	**S1 = py.textwrap.wrap('This is a string');** **import py.textwrap.wrap** **S2 = wrap('another string');**
import mymod as mm	**mm = py.importlib.import_module('mymod');** **% Use mm as an alias to access functionality in mymod**

Python 函数的帮助

有关 Python 功能的完整说明，请咨询外部资源，尤其是 https://www.python.org。Python 文档有不同版本，请务必参考与您系统上的版本对应的文档版本。MATLAB 文档中的许多示例引用 Python 标准库中的函数。

要使用第三方或用户定义的 Python 模块中的函数，请参阅您的供应商产品文档，了解有关如何安装该模块的信息以及有关其功能的详细信息。

MATLAB **py.help** 命令显示 **www.python.org/doc** 上提供的 Python 帮助。包和类的帮助可能包含很多内容，在 MATLAB 命令行窗口中显示时可能不是很有用。

- 包

  ```
  py.help('textwrap')
  ```

- 类

  ```
  py.help('textwrap.TextWrapper')
  ```

- 类的方法

  ```
  py.help('textwrap.TextWrapper.wrap')
  ```

- 函数

  ```
  py.help('textwrap.fill')
  ```

如果 MATLAB 显示以 **Python Error:** 开头的错误消息，请参考您的 Python 文档以了解详细信息。

注意 Tab 键自动填充不显示可用的 Python 功能。

您无法在 MATLAB 中使用交互式 Python 帮助 - 调用不带输入参数的 **py.help**。

使用 MATLAB 调用 Python 方法时发生名称冲突

如果 Python 方法名称是 MATLAB 基类或保留函数的 Sealed 方法的名称，则 MATLAB 会重命名该方法。新名称以字母 **x** 开头，并将原始名称的第一个字母改为大写。例如，MATLAB 将 Python 方法 **cat** 重命名为 **xCat**。有关保留方法的列表，请参阅“Methods That Modify Default Behavior”。

如果方法名称是 MATLAB 关键字，则 MATLAB 调用 **matlab.lang.makeValidName** 来重命名该方法。有关关键字列表，请参阅 **iskeyword**。

如果生成的名称是重复名称，则 MATLAB 使用 **matlab.lang.makeUniqueStrings** 重命名该方法。

调用 Python eval 函数

此示例说明如何使用 Python **eval** 命令计算表达式 **x+y**。阅读 **eval** 的帮助。

```
py.help('eval')
```

```
Help on built-in function eval in module __builtin__:

eval(...)
    eval(source[, globals[, locals]]) -> value

    Evaluate the source in the context of globals and locals.
    The source may be a string representing a Python expression
    or a code object as returned by compile().
    The globals must be a dictionary and locals can be any mapping,
    defaulting to the current globals and locals.
    If only globals is given, locals defaults to it.
```

要计算表达式，请为 **globals** 命名空间参数传递一个 Python **dict** 值。

为 **x** 和 **y** 值创建一个 Python **dict** 变量。

```
workspace = py.dict(pyargs('x',1,'y',6))
```

```
workspace =
  Python dict with no properties.

    {'y': 6.0, 'x': 1.0}
```

计算该表达式。

```
res = py.eval('x+y',workspace)
```

```
res = 7
```

或者，要添加两个数值而不分配变量，请为 **globals** 参数传递一个空的 **dict** 值。

```
res = py.eval('1+6',py.dict)
```

```
res = 7
```

执行可调用的 Python 对象

要执行可调用的 Python 对象，请使用 **feval** 函数。例如，如果 Python 类的实例 **obj** 是可调用的，请用以下 MATLAB 语句之一替换 Python 语法 **obj(x1, ..., xn)**：

```
feval(obj,x1, ..., xn)
```

```
obj(x1, ..., xn)
```

MATLAB 如何表示 Python 运算符

MATLAB 支持以下重载运算符。

Python 运算符符号	Python 方法	MATLAB 方法
+（二元）	__add__、__radd__	plus、+
-（二元）	__sub__、__rsub__	minus、-
*（二元）	__mul__、__rmul__	mtimes、*
/	__truediv__、__rtruediv__	mrdivide、/
==	__eq__	eq、==
>	__gt__	gt、>
<	__lt__	lt、<
!=	__ne__	ne、~=
>=	__ge__	ge、>=
<=	__le__	le、<=
-（一元）	__neg__	uminus、-a
+（一元）	__pos__	uplus、+a

不支持以下 Python 运算符。

Python 运算符符号	Python 方法
%	__mod__、__rmod__
**	__pow__、__rpow__
<<	__lshift__、__rlshift__
>>	__rshift__、__rrshift__
&	__and__、__rand__
^	__xor__、__rxor__
|	__or__、__ror__
//（二元）	__floordiv__、__rfloordiv__
+=（一元）	__iadd__
-=（一元）	__isub__
*=（一元）	__imul__
/=（一元）	__itruediv__
//=（一元）	__ifloordiv__
%=（一元）	__imod__
**=（一元）	__ipow__
<<=（一元）	__ilshift__
>>=（一元）	__irshift__
&=（一元）	__iand__
^=（一元）	__ixor__
!=（一元）	__ior__
~（一元）	__invert__

另请参阅

import | **feval**

详细信息

- “Handle Python Exceptions”

外部网站

- **www.python.org**

直接从 MATLAB 调用 Python 功能

您可以从 Python 库中调用功能，或直接从 MATLAB 中执行 Python 语句。

访问 Python 模块

要访问 Python 库，请在 Python 名称前添加 **py.** 前缀。例如：

```
py.list({'This','is a','list'})     % Call built-in function list
py.textwrap.wrap('This is a string') % Call wrap function in module textwrap
```

有关详细信息，请参阅“从 MATLAB 访问 Python 模块 - 快速入门” （第 21-2 页）。

运行 Python 代码

要在 MATLAB 命令提示符下执行 Python 解释器中的 Python 语句，请使用 **pyrun** 函数。使用此函数，您可以运行将 MATLAB 类型作为输入传递的代码，并将一些或所有变量返回到 MATLAB。例如，假设您在 Python 解释器中运行这条语句。

```
>>> l = ['A', 'new', 'list']
```

要从 MATLAB 运行该语句，请使用 **pyrun**。要将结果返回到 MATLAB 变量 **myList**，请添加 **"l"** 作为 **outputs** 参数：

```
myList = pyrun("l = ['A', 'new', 'list']", "l");
```

运行 Python 脚本

要从 MATLAB 命令提示符调用 Python 脚本，请使用 **pyrunfile** 函数。您传递 MATLAB 数据并返回变量的方式与 **pyrun** 相同。例如，用以下语句创建一个 **mklist.py** 文件：

```
# Python script file mklist.py:
s = 'list'
L = ['A', 'new', s]
```

从 MATLAB 运行脚本：

```
myListFile = pyrunfile("mklist.py", "L")
```

```
myListFile =
  Python list with no properties.

    ['A', 'new', 'list']
```

访问 Python 变量

当您使用 **py.** 前缀时，MATLAB 会导入整个模块，并且可以访问 Python 代码的所有函数和类。但是，当您使用 **pyrun** 或 **pyrunfile** 函数执行 Python 代码时，如果您要访问 Python 数据，则必须使用 **outvars** 参数将 Python 对象显式返回到 MATLAB。

pyrun 和 pyrunfile 函数的限制

如果将类的实例返回到 MATLAB，则无法修改使用 **pyrun** 或 **pyrunfile** 定义的 Python 类。如果需要更改类定义，请重新启动解释器会话：

```
terminate(pyenv)
pyenv("ExecutionMode","OutOfProcess")
```

或者，为 **"InProcess"** 重新启动 MATLAB。

pyrun 和 **pyrunfile** 函数不支持具有局部变量的类，这些类由其他局部变量通过方法初始化。对于这种用法，创建一个模块并使用 **py.** 前缀访问它。

另请参阅

pyrun | **pyrunfile**

相关示例

- “从 MATLAB 访问 Python 模块 - 快速入门” (第 21-2 页)

系统命令

运行外部命令、脚本和程序

您可以使用 ! 操作符或 **system** 函数从 MATLAB 命令行中执行操作系统命令。

shell 转义函数

感叹号字符 (!) 有时也称为 Bang，是一个 shell 转义字符。! 字符指示输入行的其余内容是针对操作系统的命令。操作系统决定了您能够为命令提供的输入参数列表的最大长度。使用 !，无需退出 MATLAB 即可调用实用工具或其他可执行程序。

例如，以下代码将在 UNIX 平台上使用 vi 编辑器中打开名为 **yearlystats.m** 的文件。

```
!vi yearlystats.m
```

在外部程序完成或您退出程序后，操作系统会将控制权返回给 MATLAB。要在后台模式下运行应用程序或在单独的窗口中显示输出，请在行尾添加 **&**。

例如，以下语句将打开 Microsoft Excel 程序并将控制权返回给命令提示符，以便您能够继续运行 MATLAB 命令。

```
!excel.exe &
```

在 Windows 平台上，以下命令将在 DOS 窗口中显示结果。

```
!dir &
```

注意 要在阶乘表达式中使用感叹号，请调用 **factorial** 函数。

返回结果和状态

要运行返回结果和状态的程序，请使用 **system** 函数。

指定环境变量

要使用特定的环境变量执行操作系统命令，请将针对操作系统的所有命令包含在系统调用中。这适用于 MATLAB ! (Bang)、**system**、**dos** 和 **unix** 函数。要分隔命令：

- 在 Windows 平台上，使用 **&**（与符号）
- 在 UNIX 平台上，使用 **;**（分号）

也可以在启动 MATLAB 之前设置环境变量。

在系统路径以外运行 UNIX 程序

当包含该文件的文件夹不在对 MATLAB 可见的 UNIX 系统路径上时，您可以从 MATLAB 中运行 UNIX 程序。要查看对 MATLAB 可见的路径，请在 MATLAB 命令提示符下键入以下命令。

```
getenv('PATH')
```

您可以为当前 MATLAB 会话修改系统路径，也可以跨后续 MATLAB 会话修改系统路径，如以下主题中所述：

当前 MATLAB 会话

您可以为当前 MATLAB 会话修改系统路径。当您重新启动 MATLAB 时，该文件夹将不再位于系统路径上。

要修改系统路径，请执行以下操作之一。

- 将 MATLAB 中的当前文件夹更改为包含您要运行的程序的文件夹。
- 在命令提示符下键入以下命令。

```
path1 = getenv('PATH')
path1 = [path1 ':/usr/local/bin']
setenv('PATH', path1)
!echo $PATH
```

在当前 shell 会话中跨 MATLAB 会话

您可以在 shell 会话中修改系统路径。当您在当前 shell 会话中重新启动 MATLAB 时，该文件夹仍然位于系统路径上。但是，如果您重新启动 shell 会话，然后重新启动 MATLAB，该文件夹将不再位于系统路径上。

要在 shell 中将文件夹添加到系统路径，请执行以下操作：

1. 退出 MATLAB。
2. 根据所使用的 shell，在系统命令提示符下键入以下命令之一，其中 **myfolder** 是包含您要运行的程序的文件夹：
   - 对于 **bash** 或相关 shell：

     ```
     export PATH="$PATH:myfolder"
     ```
   - 对于 **tcsh** 或相关 shell：

     ```
     setenv PATH "${PATH}:myfolder"
     ```
3. 启动 MATLAB。
4. 在 MATLAB 命令行窗口中，键入：

   ```
   !echo $PATH
   ```

跨所有 MATLAB 会话

要跨 shell 和 MATLAB 会话修改系统路径，请按“在 MATLAB 启动文件中指定启动选项”中所述在 MATLAB 启动文件中添加以下命令。

```
path1 = getenv('PATH')
path1 = [path1 ':/usr/local/bin']
setenv('PATH', path1)
!echo $PATH
```

在 macOS 上运行 AppleScript

在 macOS 平台上，不能直接从 MATLAB 中运行 Apple AppleScript 程序。要运行 AppleScript 命令，请使用 MATLAB **unix** 或 **!** (Bang) 函数调用 Apple macOS **osascript** 函数。

另请参阅

system

设置