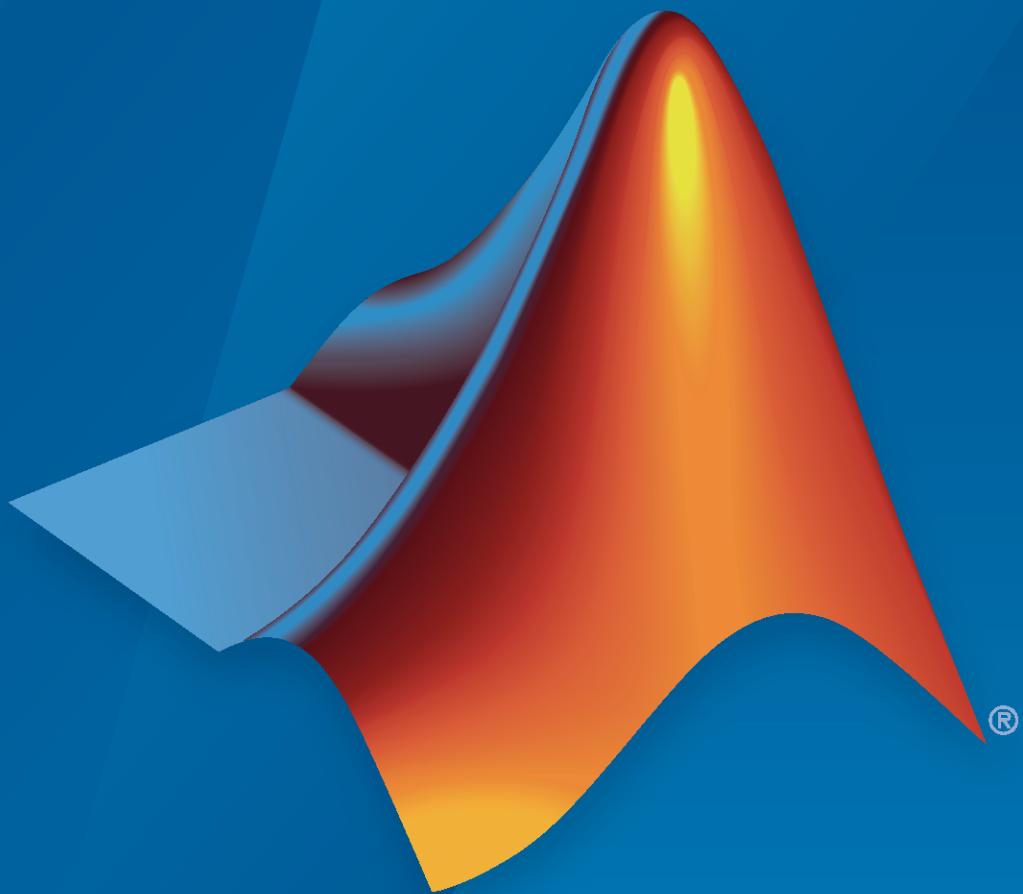


MATLAB®

数学



MATLAB®

R2020a

 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® 数学

© COPYRIGHT 1984–2020 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

2004 年 6 月	第一次印刷	MATLAB 7.0 (版本 14) 中的新增内容, 使用 MATLAB 的以前部分
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 3 月	仅限在线版本	MATLAB 7.0.4 (版本 14SP2) 中的修订内容
2005 年 6 月	第二次印刷	MATLAB 7.0.4 中的少量修订内容
2005 年 9 月	第二次印刷	MATLAB 7.1 (版本 14SP3) 中的修订内容
2006 年 3 月	第二次印刷	MATLAB 7.2 (版本 2006a) 中的修订内容
2006 年 9 月	第二次印刷	MATLAB 7.3 (版本 2006b) 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
2015 年 10 月	仅限在线版本	MATLAB 8.5.1 (版本 2015aSP1) 中的再发布内容
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

矩阵和数组

1

创建、串联和扩展矩阵	1-2
从矩阵中删除行或列	1-7
重构和重新排列数组	1-8
多维数组	1-13
数组索引	1-20

线性代数

2

MATLAB 环境中的矩阵	2-2
创建矩阵	2-2
矩阵的加法和减法	2-3
向量乘积和转置	2-3
矩阵乘法	2-5
单位矩阵	2-6
矩阵求逆	2-6
Kronecker 张量积	2-7
向量范数和矩阵范数	2-8
使用线性代数方程函数的多线程计算	2-9
线性方程组	2-10
计算注意事项	2-10
通解	2-11
方阵方程组	2-11
超定方程组	2-13
欠定方程组	2-15
多右端线性方程组的求解	2-16
迭代法	2-17
多线程计算	2-17
分解	2-19
简介	2-19
Cholesky 分解	2-19
LU 分解	2-20
QR 分解	2-21
对分解使用多线程计算	2-23

幂和指数	2-24
特征值	2-28
特征值的分解	2-28
多重特征值	2-29
Schur 分解	2-29
奇异值	2-31
MATLAB 中的 LAPACK	2-34
简史	2-34
矩阵指数	2-35
指数函数的图形比较	2-39
基本矩阵运算	2-43
判断矩阵是否为对称正定矩阵	2-50

随机数

3

随机数为什么可在启动后重复出现?	3-2
创建随机数数组	3-3
随机数函数	3-3
随机数生成器	3-3
特定范围内的随机数	3-5
随机整数	3-6
具有特定均值和方差的正态分布随机数	3-7
球体内的随机数	3-8
生成可重复的随机数	3-10
指定种子	3-10
保存和恢复生成器设置	3-11
生成不同的随机数	3-13
管理全局流	3-14
随机数的数据类型	3-16
创建和控制随机数流	3-19
子流	3-19
选择随机数生成器	3-20
多个流	3-24

更换不推荐的 rand 和 randn 语法	3-26
不建议使用的语法说明	3-26
替换语法说明	3-26
使用整数种子初始化生成器的替换语法	3-27
使用状态向量初始化生成器的替换语法	3-27
如果无法从不建议使用的语法升级	3-28
控制随机数的生成	3-29

稀疏矩阵

4

稀疏矩阵的计算优点	4-2
内存的管理	4-2
计算效率	4-2
构造稀疏矩阵	4-3
创建稀疏矩阵	4-3
导入稀疏矩阵	4-6
访问稀疏矩阵	4-7
非零元素	4-7
索引和值	4-8
稀疏矩阵运算中的索引	4-8
可视化稀疏矩阵	4-11
稀疏矩阵运算	4-13
运算效率	4-13
置换与重新排序	4-13
稀疏矩阵分解	4-16
特征值和奇异值	4-22
参考	4-24
有限差分拉普拉斯算子	4-25
稀疏矩阵的图形表示	4-29
图形和矩阵	4-35
稀疏矩阵重新排序	4-41
线性方程组的迭代方法	4-53
直接方法与迭代方法	4-53
常规迭代算法	4-53
迭代方法概要	4-54
选择迭代求解器	4-55
预条件子	4-56
均衡和重新排序	4-58
使用线性运算函数取代矩阵	4-59
参考	4-60

5

有向图和无向图	5-2
什么是图?	5-2
创建图	5-5
图节点 ID	5-8
修改或查询现有图	5-8
修改现有图的节点和边	5-10
添加图节点名称、边权重和其他属性	5-13
图的绘制和自定义	5-17
可视化广度优先搜索和深度优先搜索	5-28
使用拉普拉斯矩阵为图分区	5-33
将节点属性添加到图论图数据游标	5-36
生成 Watts-Strogatz 小世界图形模型	5-39
使用 PageRank 算法对网站进行排名	5-46
为图节点和边添加标签	5-53

6

创建并计算多项式	6-2
多项式的根	6-4
数值根	6-4
使用代换法求根	6-4
特定区间内的根	6-5
符号根	6-7
对多项式求积分和微分	6-9
多项式曲线拟合	6-11
预测美国人口	6-13
标量函数的根	6-18

三角剖分表示法	7-2
二维和三维域	7-2
三角剖分矩阵格式	7-3
使用三角剖分分类查询三角剖分	7-4
使用 Delaunay 三角剖分	7-13
Delaunay 三角剖分的定义	7-13
创建 Delaunay 三角剖分	7-14
包含重复位置的点集的三角剖分	7-34
创建和编辑 Delaunay 三角剖分	7-36
空间搜索	7-51
简介	7-51
最近邻点搜索	7-51
点位置搜索	7-53
Voronoi 图	7-57
绘图二维 Voronoi 图和 Delaunay 三角剖分	7-57
计算 Voronoi 图	7-60
区域边界的类型	7-64
凸包与非凸多边形	7-64
阿尔法形状	7-66
计算凸包	7-70
使用 convhull 和 convhulln 计算凸包	7-70
使用 delaunayTriangulation 类计算凸包	7-73
使用 alphaShape 的凸包计算	7-74

网格和散点样本数据	8-2
插入网格数据	8-3
网格数据表示	8-3
基于网格的插值	8-10
interp 系列函数的插值	8-15
用 griddedInterpolant 类插值	8-24
多个一维值集的插值	8-33
将二维选择项插入三维网格中	8-35
内插散点数据	8-37
散点数据	8-37
使用 griddata 和 griddatan 插入散点数据	8-39

scatteredInterpolant 类	8-42
使用 scatteredInterpolant 类插入散点数据	8-43
复数散点数据的插值	8-49
解决散点数据插值中的问题	8-51
使用特定 Delaunay 三角剖分的插值	8-59
使用 delaunayTriangulation 查询的最近邻点插值	8-59
使用 delaunayTriangulation 查询的线性插值	8-59
外插散点数据	8-62
影响外插准确性的因素	8-62
比较粗略采样和精细采样的散点数据的外插	8-62
三维数据外插	8-66
对不同量级的数据进行归一化	8-68
使用网格插值对图像重采样	8-71

优化

9

优化非线性函数	9-2
求一元函数的最小值	9-2
求多元函数的最小值	9-3
求函数最大值	9-3
fminsearch 算法	9-4
参考	9-5
通过优化拟合曲线	9-6
设置优化选项	9-8
如何设置选项	9-8
选项表	9-8
容差和终止条件	9-9
输出结构体	9-9
优化求解器迭代显示	9-11
优化求解器输出函数	9-12
什么是输出函数?	9-12
创建和使用输出函数	9-12
输出函数的结构体	9-13
嵌套输出函数的示例	9-13
optimValues 中的字段	9-15
算法的状态	9-15
Stop 标签	9-16
优化求解器绘制函数	9-17
什么是绘图函数?	9-17
示例：绘图函数	9-17
优化故障排除和提示	9-19

参数化函数	10-2
概述	10-2
使用嵌套函数参数化	10-2
使用匿名函数进行参数化	10-2

常微分方程 (ODE)

选择 ODE 求解器	11-2
常微分方程	11-2
ODE 的类型	11-2
ODE 方程组	11-2
高阶 ODE	11-3
复数 ODE	11-4
基本求解器选择	11-4
ODE 示例和文件摘要	11-6
ODE 选项摘要	11-9
选项语法	11-9
选项与每个求解器的兼容性	11-9
用法示例	11-10
ODE 事件位置	11-11
什么是事件位置?	11-11
编写事件函数	11-11
事件信息	11-12
局限性	11-12
简单事件位置: 弹球	11-12
高级事件位置: 限制性三体问题	11-13
求解非刚性 ODE	11-17
解算刚性 ODE	11-21
解算微分代数方程 (DAE)	11-27
什么是微分代数方程?	11-27
一致的初始条件	11-28
微分指数	11-28
施加非负性	11-28
将 Robertson 问题作为半隐式微分代数方程 (DAE) 求解	11-29
非负 ODE 解	11-32
常见 ODE 问题疑难解答	11-35
误差容限	11-35
问题规模	11-35
解分量	11-36
问题类型	11-37

微分方程	11-39
求解捕食者-猎物方程	11-48

边界值问题 (BVP)

12

求解边界值问题	12-2
边界条件	12-2
解的初始估计值	12-2
查找未知参数	12-3
奇异 BVP	12-3
BVP 求解器的选择	12-4
解的计算	12-4
BVP 示例和文件	12-4
对具有两个解的 BVP 求解	12-7
求解具有未知参数的 BVP	12-10
使用延拓求解 BVP 问题	12-14
使用延拓验证 BVP 一致性	12-18
求解具有奇异项的 BVP	12-23
求解具有多边界条件的 BVP	12-27

偏微分方程 (PDE)

13

求解偏微分方程	13-2
使用 MATLAB 可求解哪些类型的 PDE?	13-2
求解一维 PDE	13-2
示例：热方程	13-4
PDE 示例和文件	13-7
求解单个 PDE	13-9
求解具有不连续性的 PDE	13-16
求解 PDE 并计算偏导数	13-22
求解 PDE 方程组	13-30
使用初始条件阶跃函数求解 PDE 方程组	13-37

14

解算时滞微分方程	14-2
常时滞 DDE	14-2
时间相关和状态相关的 DDE	14-2
中立型 DDE	14-2
计算特定点的解	14-2
历史解和初始值	14-3
DDE 中的不连续性	14-3
不连续性传播	14-3
DDE 示例和文件	14-3
具有常时滞的 DDE	14-5
具有状态相关时滞的 DDE	14-8
具有不连续性的血管模型 DDE	14-12
中立型 DDE	14-17
中立型的初始值 DDE	14-21

数值积分**15**

计算弧线长度的积分	15-2
复曲线积分	15-3
积分域内部的奇点	15-5
多项式积分的解析解	15-7
数值数据的积分	15-8
计算表面的切平面	15-12

傅里叶变换**16**

傅里叶变换	16-2
基本频谱分析	16-8
频谱分析数量	16-8
含噪信号	16-8
音频信号	16-10

使用 FFT 进行多项式插值	16-14
数学中的 FFT	16-14
小行星数据插值	16-14
二维傅里叶变换	16-17
二维傅里叶变换	16-17
二维衍射模式	16-17
使用 FFT 进行频谱分析	16-21
从正弦波转换为方波	16-24
使用 FFT 分析周期性数据	16-29

矩阵和数组

- “创建、串联和扩展矩阵” (第 1-2 页)
- “从矩阵中删除行或列” (第 1-7 页)
- “重构和重新排列数组” (第 1-8 页)
- “多维数组” (第 1-13 页)
- “数组索引” (第 1-20 页)

创建、串联和扩展矩阵

最基本的 MATLAB® 数据结构体是矩阵。矩阵是按行和列排列的数据元素的二维矩形数组。元素可以是数字、逻辑值（`true` 或 `false`）、日期和时间、字符串或者其他 MATLAB 数据类型。

即使一个数字也能以矩阵的形式存储。例如，包含值 100 的变量存储为 `double` 类型的 1×1 矩阵。

```
A = 100;
whos A
```

Name	Size	Bytes	Class	Attributes
A	1×1	8	double	

构建数据矩阵

如果您有一组具体的数据，可以使用方括号将这些元素排列成矩阵。一行数据的元素之间用空格或逗号分隔，行与行之间用分号分隔。例如，创建只有一行的矩阵，其中包含四个数字元素。得到的矩阵大小为 1×4 ，因为它有一行和四列。这种形状的矩阵通常称为行向量。

```
A = [12 62 93 -8]
```

```
A = 1×4
```

```
12 62 93 -8
```

```
sz = size(A)
```

```
sz = 1×2
```

```
1 4
```

现在再用这些数字创建一个矩阵，但排成两行。此矩阵有两行和两列。

```
A = [12 62; 93 -8]
```

```
A = 2×2
```

```
12 62
93 -8
```

```
sz = size(A)
```

```
sz = 1×2
```

```
2 2
```

专用矩阵函数

MATLAB 中有许多函数可以帮助您创建具有特定值或特定结构的矩阵。例如，`zeros` 和 `ones` 函数可以创建元素全部为零或全部为一的矩阵。这些函数的第一个和第二个参数分别是矩阵的行数和列数。

```
A = zeros(3,2)
```

```
A = 3×2
```

```
0  0
0  0
0  0
```

B = ones(2,4)

B = 2×4

```
1  1  1  1
1  1  1  1
```

diag 函数将输入元素放在矩阵的对角线上。例如，创建一个行向量 A，其中包含四个元素。然后创建一个 4×4 矩阵，其对角元素是 A 的元素。

```
A = [12 62 93 -8];
B = diag(A)
```

B = 4×4

```
12  0  0  0
0  62  0  0
0  0  93  0
0  0  0  -8
```

串联矩阵

您还可以使用方括号将现有矩阵连接在一起。这种创建矩阵的方法称为串联。例如，将两个行向量串联起来，形成一个更长的行向量。

```
A = ones(1,4);
B = zeros(1,4);
C = [A B]
```

C = 1×8

```
1  1  1  1  0  0  0  0
```

要将 A 和 B 排列为一个矩阵的两行，请使用分号。

D = [A;B]

D = 2×4

```
1  1  1  1
0  0  0  0
```

要串联两个矩阵，它们的大小必须兼容。也就是说，水平串联矩阵时，它们的行数必须相同。垂直串联矩阵时，它们的列数必须相同。例如，水平串联两个各自包含两行的矩阵。

A = ones(2,3)

A = 2×3

```
1  1  1
```

```
1   1   1
```

B = zeros(2,2)

B = 2×2

```
0   0  
0   0
```

C = [A B]

C = 2×5

```
1   1   1   0   0  
1   1   1   0   0
```

串联矩阵的另一种方法是使用串联函数，如 **horzcat**，它可以水平串联两个兼容的输入矩阵。

D = horzcat(A,B)

D = 2×5

```
1   1   1   0   0  
1   1   1   0   0
```

生成数值序列

colon 是创建元素连续且均匀分布的矩阵的便捷方式。例如，创建一个行向量，其元素是从 1 到 10 的整数。

A = 1:10

A = 1×10

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

可以使用冒号运算符创建在任何范围内以 1 为增量的数字序列。

A = -2.5:2.5

A = 1×6

```
-2.5000  -1.5000  -0.5000  0.5000  1.5000  2.5000
```

要更改序列的增量值，请在范围起始值和结束值之间指定增量值，以冒号分隔。

A = 0:2:10

A = 1×6

```
0   2   4   6   8   10
```

要递减，请使用负数。

```
A = 6:-1:0
```

A = 1×7

```
6 5 4 3 2 1 0
```

还可以按非整数值递增。如果增量值不能均分指定的范围，MATLAB 会在超出范围之前在可以达到的最后一个值处自动结束序列。

```
A = 1:0.2:2.1
```

A = 1×6

```
1.0000 1.2000 1.4000 1.6000 1.8000 2.0000
```

扩展矩阵

通过将一个或多个元素置于现有行和列索引边界之外，可以将它们添加到矩阵中。MATLAB 会自动用 0 填充矩阵，使其保持为矩形。例如，创建一个 2×3 矩阵，然后在 (3,4) 的位置插入一个元素，使矩阵增加一行一列。

```
A = [10 20 30; 60 70 80]
```

A = 2×3

```
10 20 30
60 70 80
```

A(3,4) = 1

A = 3×4

```
10 20 30 0
60 70 80 0
0 0 0 1
```

还可以通过在现有索引范围之外插入新矩阵来扩展其大小。

```
A(4:5,5:6) = [2 3; 4 5]
```

A = 5×6

```
10 20 30 0 0 0
60 70 80 0 0 0
0 0 0 1 0 0
0 0 0 0 2 3
0 0 0 0 4 5
```

要重复扩展矩阵的大小，例如在 **for** 循环中，通常最好要为预计创建的最大矩阵预分配空间。如果没有预分配空间，MATLAB 必须在每次大小增加时分配内存，因此会降低操作速度。例如，通过将矩阵的元素初始化为零，预分配一个最多容纳 10000 行和 10000 列的矩阵。

```
A = zeros(10000,10000);
```

如果之后还要预分配更多元素，可以通过在矩阵索引范围之外指定元素或将另一个预分配的矩阵与 A 串联来进行扩展。

空数组

MATLAB 中的空数组是指至少有一个维度的长度等于零的数组。空数组可用于以编程方式表示“无”的概念。例如，假设要查找一个向量中小于 0 的所有元素但没有找到。`find` 函数将返回一个空的索引向量，表示未找到任何小于 0 的元素。

```
A = [1 2 3 4];  
ind = find(A<0)
```

```
ind =
```

```
1x0 empty double row vector
```

许多算法都包含可以返回空数组的函数调用。在这些算法中，允许将空数组作为函数参数传递，而不是作为特殊情况处理，这样通常是有帮助的。如果确实需要自定义空数组的处理方式，可以使用 `isempty` 函数检查它们。

```
TF = isempty(ind)
```

```
TF = logical  
1
```

另请参阅

相关示例

- “数组索引”（第 1-20 页）
- “重构和重新排列数组”（第 1-8 页）
- “多维数组”（第 1-13 页）
- “创建字符串数组”
- “表示 MATLAB 中的日期和时间”

从矩阵中删除行或列

要删除矩阵的行或列，最简单的方法是将该行或列设置为等于空方括号 []。例如，创建一个 4×4 矩阵并删除第二行。

```
A = magic(4)
```

```
A = 4×4
```

```
16   2   3   13  
5   11  10   8  
9   7   6   12  
4   14  15   1
```

```
A(2,:) = []
```

```
A = 3×4
```

```
16   2   3   13  
9   7   6   12  
4   14  15   1
```

现在删除第三列。

```
A(:,3) = []
```

```
A = 3×3
```

```
16   2   13  
9   7   12  
4   14  1
```

此方法可以扩展到任何数组。例如，创建一个随机的 $3 \times 3 \times 3$ 数组，然后删除第三维第一个矩阵中的所有元素。

```
B = rand(3,3,3);  
B(:,:,1) = [];
```

另请参阅

相关示例

- “重构和重新排列数组”（第 1-8 页）
- “数组索引”（第 1-20 页）

重构和重新排列数组

MATLAB® 中的许多函数都可以提取现有数组的元素，然后按照不同的形状或顺序放置。这样有助于预处理数据，便于之后进行计算或分析。

重构

`reshape` 函数可以更改数组的大小和形状。例如，将 3×4 矩阵重构为 2×6 矩阵。

```
A = [1 4 7 10; 2 5 8 11; 3 6 9 12]
```

$A = 3 \times 4$

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

```
B = reshape(A,2,6)
```

$B = 2 \times 6$

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

只要不同形状中的元素数量相同，就可以将它们重构为具有任意维度的数组。使用 A 中的元素，创建一个 $2 \times 2 \times 3$ 的多维数组。

```
C = reshape(A,2,2,3)
```

```
C =
C(:,:,1) =
```

1	3
2	4

```
C(:,:,2) =
```

5	7
6	8

```
C(:,:,3) =
```

9	11
10	12

转置和翻转

线性代数中常见的任务是转置矩阵，即将矩阵的行变成列，将列变成行。要实现此目的，可以使用 `transpose` 函数或 `.'` 运算符。

创建一个 3×3 矩阵并计算其转置。

```
A = magic(3)
```

A = 3×3

8	1	6
3	5	7
4	9	2

B = A.'

B = 3×3

8	3	4
1	5	9
6	7	2

类似的运算符'可以计算复矩阵的共轭转置。此操作计算每个元素的复共轭并对其进行转置。创建一个 2×2 复矩阵并计算其共轭转置。

A = [1+i 1-i; -i i]

A = 2×2 complex

1.0000 + 1.0000i	1.0000 - 1.0000i
0.0000 - 1.0000i	0.0000 + 1.0000i

B = A'

B = 2×2 complex

1.0000 - 1.0000i	0.0000 + 1.0000i
1.0000 + 1.0000i	0.0000 - 1.0000i

flipud 上下翻转矩阵的行, **fliplr** 左右翻转矩阵的列。

A = [1 2; 3 4]

A = 2×2

1	2
3	4

B = flipud(A)

B = 2×2

3	4
1	2

C = fliplr(A)

C = 2×2

2	1
4	3

平移和旋转

使用 `circshift` 函数，可以将数组的元素平移一定的位置数。例如，创建一个 3×4 矩阵，然后将其各列向右平移 2 个位置。第二个参数 [0 2] 要求 `circshift` 将各行平移 0 个位置，将各列向右平移 2 个位置。

```
A = [1 2 3 4; 5 6 7 8; 9 10 11 12]
```

$A = 3 \times 4$

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

```
B = circshift(A,[0 2])
```

$B = 3 \times 4$

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

要将 A 的各行向上平移 1 个位置而各列保持不动，请将第二个参数指定为 [-1 0]。

```
C = circshift(A,[-1 0])
```

$C = 3 \times 4$

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

`rot90` 函数可以将矩阵逆时针旋转 90 度。

```
A = [1 2; 3 4]
```

$A = 2 \times 2$

1	2
3	4

```
B = rot90(A)
```

$B = 2 \times 2$

2	4
1	3

如果再旋转 3 次（使用第二个参数指定旋转次数），最后将得到原始矩阵 A。

```
C = rot90(B,3)
```

$C = 2 \times 2$

1	2
---	---

```
3 4
```

排序

对数组中的数据进行排序也是一项实用功能，MATLAB 提供了几种排序方法。例如，**sort** 函数可以按升序或降序对矩阵的每一行或每一列中的元素进行排序。创建矩阵 A，并按升序对 A 的每一列进行排序。

```
A = magic(4)
```

```
A = 4×4
```

```
16 2 3 13  
5 11 10 8  
9 7 6 12  
4 14 15 1
```

```
B = sort(A)
```

```
B = 4×4
```

```
4 2 3 1  
5 7 6 8  
9 11 10 12  
16 14 15 13
```

按降序对每一行进行排序。第二个参数值 2 指定您希望逐行排序。

```
C = sort(A,2,'descend')
```

```
C = 4×4
```

```
16 13 3 2  
11 10 8 5  
12 9 7 6  
15 14 4 1
```

要整行排序，请使用 **sortrows** 函数。例如，根据第一列中的元素按升序对 A 的各行排序。行的位置发生变化，但每一行中元素的顺序不变。

```
D = sortrows(A)
```

```
D = 4×4
```

```
4 14 15 1  
5 11 10 8  
9 7 6 12
```

16 2 3 13

另请参阅

相关示例

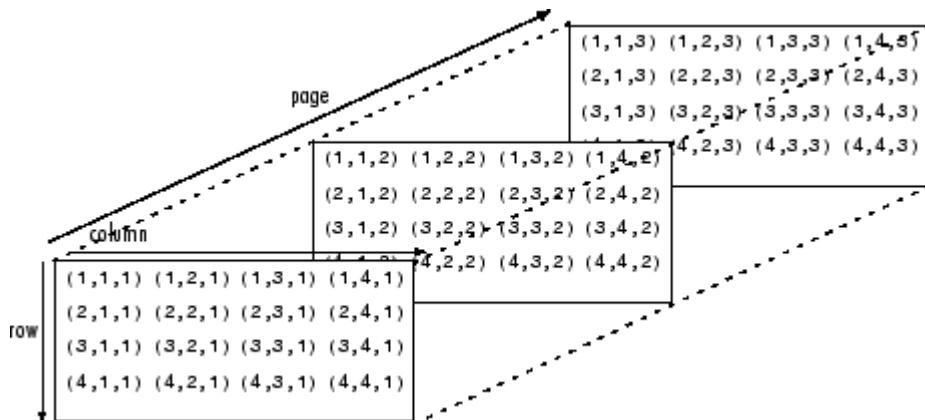
- “从矩阵中删除行或列” (第 1-7 页)
- “数组索引” (第 1-20 页)

多维数组

MATLAB® 中的多维数组是指具有两个以上维度的数组。在矩阵中，两个维度由行和列表示。

		column			
		(1,1)	(1,2)	(1,3)	(1,4)
row	(2,1)	(2,2)	(2,3)	(2,4)	
	(3,1)	(3,2)	(3,3)	(3,4)	
	(4,1)	(4,2)	(4,3)	(4,4)	

每个元素由两个下标（即行索引和列索引）来定义。多维数组是二维矩阵的扩展，并使用额外的下标进行索引。例如，三维数组使用三个下标。前两个维度就像一个矩阵，而第三个维度表示元素的页数或张数。



创建多维数组

要创建多维数组，可以先创建二维矩阵，然后再进行扩展。例如，首先定义一个 3×3 矩阵，作为三维数组中的第一页。

`A = [1 2 3; 4 5 6; 7 8 9]`

`A = 3×3`

```

1   2   3
4   5   6
7   8   9

```

现在添加第二页。要完成此操作，可将另一个 3×3 矩阵赋给第三个维度中的索引值 2。语法 `A(:,:,2)` 在第一个和第二个维度中使用冒号，以在其中包含赋值表达式右侧的所有行和所有列。

`A(:,:,2) = [10 11 12; 13 14 15; 16 17 18]`

```

A =
A(:,:,1) =

```

```

1   2   3
4   5   6
7   8   9

```

A(:,:,2) =

```
10 11 12  
13 14 15  
16 17 18
```

cat 函数可用于构造多维数组。例如，在 A 后以串联方式添加第三页，由此创建一个新的三维数组 B。第一个参数指示要沿哪一个维度进行串联。

B = cat(3,A,[3 2 1; 0 9 8; 5 3 7])

B =
B(:,:,1) =

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

B(:,:,2) =

```
10 11 12  
13 14 15  
16 17 18
```

B(:,:,3) =

```
3 2 1  
0 9 8  
5 3 7
```

快速扩展多维数组的另一种方法是将一个元素赋给一整页。例如，为数组 B 添加第四页，其中包含的值全部为零。

B(:,:,4) = 0

B =
B(:,:,1) =

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

B(:,:,2) =

```
10 11 12  
13 14 15  
16 17 18
```

B(:,:,3) =

```
3  2  1
0  9  8
5  3  7
```

B(:, :, 4) =

```
0  0  0
0  0  0
0  0  0
```

访问元素

要访问多维数组中的元素，请使用整数下标，就像在向量和矩阵中一样。例如，找到 A 中下标为 1,2,2 的元素，它位于 A 的第二页上的第一行第二列。

A

A =
A(:, :, 1) =

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

A(:, :, 2) =

```
10 11 12
13 14 15
16 17 18
```

elA = A(1, 2, 2)

elA = 11

在第二个维度中使用索引向量 [1 3]，只访问 A 的每一页上的第一列和最后一列。

C = A(:, [1 3], :)

C =
C(:, :, 1) =

```
1  3
4  6
7  9
```

C(:, :, 2) =

```
10 12
13 15
16 18
```

要查找每一页的第二行和第三行，请使用冒号运算符创建索引向量。

```
D = A(2:3,:,:)
```

```
D =  
D(:,:,1) =
```

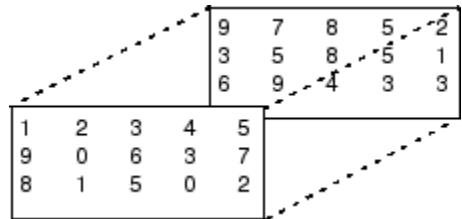
```
4 5 6  
7 8 9
```

```
D(:,:,2) =
```

```
13 14 15  
16 17 18
```

操作数组

多维数组的元素可以通过多种方式移动，类似于向量和矩阵。**reshape**、**permute** 和 **squeeze** 函数可用于重新排列元素。假设有一个两页的三维数组。



重构多维数组有助于执行某些操作或可视化数据。使用 **reshape** 函数，将一个三维数组的元素重新排列成 6×5 矩阵。

```
A = [1 2 3 4 5; 9 0 6 3 7; 8 1 5 0 2];  
A(:,:,2) = [9 7 8 5 2; 3 5 8 5 1; 6 9 4 3 3];  
B = reshape(A,[6 5])
```

B = 6×5

```
1 3 5 7 5  
9 6 7 5 5  
8 5 2 9 3  
2 4 9 8 2  
0 3 3 8 1  
1 0 6 4 3
```

reshape 逐列操作，沿 A 中各列连续逐一提取元素来创建新矩阵，从第一页开始，之后是第二页。

置换操作用于重新排列数组的维度顺序。假设有一个三维数组 M。

```
M(:,:,1) = [1 2 3; 4 5 6; 7 8 9];  
M(:,:,2) = [0 5 4; 2 7 6; 9 3 1]
```

```
M =  
M(:,:,1) =
```

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

```
M(:,:,2) =
```

0	5	4
2	7	6
9	3	1

使用 **permute** 函数，通过在第二个参数中指定维度顺序，将每一页上的行下标和列下标交换。**M** 的原始行现在是列，原始列现在是行。

```
P1 = permute(M,[2 1 3])
```

```
P1 =
```

```
P1(:,:,1) =
```

1	4	7
2	5	8
3	6	9

```
P1(:,:,2) =
```

0	2	9
5	7	3
4	6	1

同样，将 **M** 的行下标和页下标交换。

```
P2 = permute(M,[3 2 1])
```

```
P2 =
```

```
P2(:,:,1) =
```

1	2	3
0	5	4

```
P2(:,:,2) =
```

4	5	6
2	7	6

```
P2(:,:,3) =
```

7	8	9
9	3	1

操作多维数组时，您可能会遇到某些数组有一个长度为 1 的多余维度。**squeeze** 函数可以执行另一种操作，消除长度为 1 的维度。例如，使用 **repmat** 函数创建一个 $2 \times 3 \times 1 \times 4$ 数组，其元素全部为 5，第三个维度的长度为 1。

```
A = repmat(5,[2 3 1 4])
```

```
A =
```

```
A(:,:,1,1) =
```

```
5   5   5  
5   5   5
```

```
A(:,:,1,2) =
```

```
5   5   5  
5   5   5
```

```
A(:,:,1,3) =
```

```
5   5   5  
5   5   5
```

```
A(:,:,1,4) =
```

```
5   5   5  
5   5   5
```

```
szA = size(A)
```

```
szA = 1×4
```

```
2   3   1   4
```

```
numdimsA = ndims(A)
```

```
numdimsA = 4
```

使用 **squeeze** 函数删除第三个维度，从而得到一个三维数组。

```
B = squeeze(A)
```

```
B =
```

```
B(:,:,1) =
```

```
5   5   5  
5   5   5
```

```
B(:,:,2) =
```

```
5   5   5  
5   5   5
```

```
B(:,:,3) =
```

```
5   5   5  
5   5   5
```

```
B(:,:,4) =
```

```
5   5   5  
5   5   5
```

```
szB = size(B)
```

```
szB = 1×3
```

```
2   3   4
```

```
numdimsB = ndims(B)
```

```
numdimsB = 3
```

另请参阅

相关示例

- “创建、串联和扩展矩阵” (第 1-2 页)
- “数组索引” (第 1-20 页)
- “重构和重新排列数组” (第 1-8 页)

数组索引

在 MATLAB® 中，根据元素在数组中的位置（索引）访问数组元素的方法主要有三种：按位置索引、线性索引和逻辑索引。

按元素位置进行索引

最常见的方法是显式指定元素的索引。例如，要访问矩阵中的某个元素，请依序指定该元素的行号和列号。

```
A = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]
```

```
A = 4×4
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

```
e = A(3,2)
```

```
e = 10
```

e 是 A 中位于 3,2 位置（第三行第二列）的元素。

您还可以在一个向量中指定多个元素的索引，从而一次引用多个元素。例如，访问 A 的第二行中的第一个和第三个元素。

```
r = A(2,[1 3])
```

```
r = 1×2
```

5	7
---	---

要访问某个行范围或列范围内的元素，请使用 **colon**。例如，访问 A 中第一到三行、第二到四列中的元素。

```
r = A(1:3,2:4)
```

```
r = 3×3
```

2	3	4
6	7	8
10	11	12

计算 r 的另一种方法是使用关键字 **end** 指定第二直至最后一列。通过此方法，您可以直接指定最后一列，而不必知道 A 中到底有多少列。

```
r = A(1:3,2:end)
```

```
r = 3×3
```

2	3	4
6	7	8
10	11	12

如果要访问所有行或所有列，只使用冒号运算符即可。例如，返回 A 的整个第三列。

```
r = A(:,3)
```

```
r = 4×1
```

```
3  
7  
11  
15
```

通常，您可以使用索引来访问 MATLAB 中任何数组的元素，而不管其数据类型或维度如何。例如，直接访问 **datetime** 数组的列。

```
t = [datetime(2018,1:5,1); datetime(2019,1:5,1)]
```

```
t = 2x5 datetime
```

```
01-Jan-2018 01-Feb-2018 01-Mar-2018 01-Apr-2018 01-May-2018  
01-Jan-2019 01-Feb-2019 01-Mar-2019 01-Apr-2019 01-May-2019
```

```
march1 = t(:,3)
```

```
march1 = 2x1 datetime
```

```
01-Mar-2018  
01-Mar-2019
```

对于更高维度的数组，可以扩展语法以匹配数组维度。假设有一个随机的 $3 \times 3 \times 3$ 数值数组。访问位于该数组第一页中第二行第三列的元素。

```
A = rand(3,3,3);  
e = A(2,3,1)
```

```
e = 0.5469
```

有关操作多维数组的详细信息，请参阅“[多维数组](#)”（第 1-13 页）。

使用单个索引进行索引

访问数组元素的另一种方法是只使用单个索引，而不管数组的大小或维度如何。此方法称为线性索引。虽然 MATLAB 根据定义的大小和形状显示数组，但实际上数组在内存中都存储为单列元素。我们可以使用矩阵来直观地理解这一概念。下面的数组虽然显示为 3×3 矩阵，但 MATLAB 将它存储为单列，由 A 的各列顺次连接而成。存储的向量包含由元素 12、45、33、36、29、25、91、48、11 组成的序列，可以用单个冒号全部显示。

```
A = [12 36 91; 45 29 48; 33 25 11]
```

```
A = 3×3
```

```
12 36 91  
45 29 48  
33 25 11
```

```
Alinear = A(:)
```

```
Alinear = 9×1
```

```
12  
45  
33  
36  
29  
25  
91  
48  
11
```

例如，**A** 的第 3,2 个元素是 **25**，您可以使用语法 **A(3,2)** 访问它。您也可以使用语法 **A(6)** 访问此元素，因为 **25** 是存储的向量序列中的第六个元素。

```
e = A(3,2)  
e = 25  
elinear = A(6)  
elinear = 25
```

线性索引在视觉上可能不太直观，但在执行某些不依赖于数组大小或形状的计算时很有用。例如，您可以轻松地对 **A** 的所有元素求和，而无需指定 **sum** 函数的第二个参数。

```
s = sum(A(:))  
s = 330
```

sub2ind 和 **ind2sub** 函数可用于在数组的原始索引和线性索引之间进行转换。例如，计算 **A** 的第 3,2 个元素的线性索引。

```
linearidx = sub2ind(size(A),3,2)  
linearidx = 6
```

从线性索引转换回行和列形式。

```
[row,col] = ind2sub(size(A),6)  
row = 3  
col = 2
```

使用逻辑值进行索引

使用 **true** 和 **false** 逻辑指示符也可以对数组进行索引，在处理条件语句时尤其便利。例如，假设您想知道矩阵 **A** 中的元素是否小于另一个矩阵 **B** 中的对应元素。当 **A** 中的元素小于 **B** 中的对应元素时，小于号运算符返回元素为 **1** 的逻辑数组。

```
A = [1 2 6; 4 3 6]  
A = 2×3
```

```
1   2   6  
4   3   6
```

```
B = [0 3 7; 3 7 5]
```

```
B = 2×3
```

```
0   3   7
3   7   5
```

```
ind = A<B
```

```
ind = 2x3 logical array
```

```
0 1 1
0 1 0
```

现在已经知道满足条件的元素的位置，可以使用 **ind** 作为索引数组来检查各个值。MATLAB 将 **ind** 中值 1 的位置与 **A** 和 **B** 中的对应元素进行匹配，并在列向量中列出它们的值。

```
Avals = A(ind)
```

```
Avals = 3×1
```

```
2
3
6
```

```
Bvals = B(ind)
```

```
Bvals = 3×1
```

```
3
7
7
```

MATLAB 中的 **is** 函数还返回逻辑数组，指示输入中的哪些元素满足特定条件。例如，使用 **ismissing** 函数检查 **string** 向量中的哪些元素是缺失值。

```
str = ["A" "B" missing "D" "E" missing];
ind = ismissing(str)
```

```
ind = 1x6 logical array
```

```
0 0 1 0 0 1
```

假设要查找非缺失值元素的值。将 \sim 运算符和索引向量 **ind** 结合使用即可实现此目的。

```
strvals = str(~ind)
```

```
strvals = 1x4 string
"A"  "B"  "D"  "E"
```

有关使用逻辑索引的更多示例，请参阅“[查找符合条件的数组元素](#)”。

另请参阅

相关示例

- “[使用分类数组访问数据](#)”
- “[访问表中的数据](#)”
- “[访问结构体数组中的数据](#)”
- “[访问元胞数组中的数据](#)”

线性代数

- “MATLAB 环境中的矩阵” (第 2-2 页)
- “线性方程组” (第 2-10 页)
- “分解” (第 2-19 页)
- “幂和指数” (第 2-24 页)
- “特征值” (第 2-28 页)
- “奇异值” (第 2-31 页)
- “MATLAB 中的 LAPACK” (第 2-34 页)
- “矩阵指数” (第 2-35 页)
- “指数函数的图形比较” (第 2-39 页)
- “基本矩阵运算” (第 2-43 页)
- “判断矩阵是否为对称正定矩阵” (第 2-50 页)

MATLAB 环境中的矩阵

本主题介绍如何在 MATLAB 中创建矩阵和执行基本矩阵计算。

MATLAB 环境使用术语矩阵来表示包含以二维网格排列的实数或复数的变量。更广泛而言，数组为向量、矩阵或更高维度的数值网格。MATLAB 中的所有数组都是矩形，在这种意义上沿任何维度的分量向量的长度均相同。矩阵中定义的数学运算是线性代数的主题。

创建矩阵

MATLAB 提供了许多函数，用于创建各种类型的矩阵。例如，您可以使用基于帕斯卡三角形的项创建一个对称矩阵：

```
A = pascal(3)
```

```
A =
 1   1   1
 1   2   3
 1   3   6
```

您也可以创建一个非对称幻方矩阵，它的行总和与列总和相等：

```
B = magic(3)
```

```
B =
 8   1   6
 3   5   7
 4   9   2
```

另一个示例是由随机整数构成的 3×2 矩形矩阵：在这种情况下，`randi` 的第一个输入描述整数可能值的范围，后面两个输入描述行和列的数量。

```
C = randi(10,3,2)
```

```
C =
 9   10
10    7
 2    1
```

列向量为 $m \times 1$ 矩阵，行向量为 $1 \times n$ 矩阵，标量为 1×1 矩阵。要手动定义矩阵，请使用方括号 [] 来表示数组的开始和结束。在括号内，使用分号 ; 表示行的结尾。在标量 (1×1 矩阵) 的情况下，括号不是必需的。例如，以下语句生成一个列向量、一个行向量和一个标量：

```
u = [3; 1; 4]
```

```
v = [2 0 -1]
```

```
s = 7
```

```
u =
 3
 1
 4
```

```
v =
```

```
2 0 -1
```

```
s =  
7
```

有关创建和处理矩阵的详细信息，请参阅“创建、串联和扩展矩阵”（第 1-2 页）。

矩阵的加法和减法

矩阵和数组的加减法是逐个元素执行的，或者说是按元素执行的。例如，**A** 加 **B** 之后再减去 **A** 又可以得到 **B**：

```
X = A + B
```

```
X =  
9 2 7  
4 7 10  
5 12 8
```

```
Y = X - A
```

```
Y =  
8 1 6  
3 5 7  
4 9 2
```

加法和减法要求两个矩阵具有兼容的维度。如果维度不兼容，将会导致错误：

```
X = A + C
```

```
Error using +  
Matrix dimensions must agree.
```

有关详细信息，请参阅“数组与矩阵运算”。

向量乘积和转置

长度相同的行向量和列向量可以按任一顺序相乘。其结果是一个标量（称为内积）或一个矩阵（称为外积）：

```
u = [3; 1; 4];  
v = [2 0 -1];  
x = v*u
```

```
x =
```

```
2
```

```
X = u*v
```

```
X =
```

```
6 0 -3  
2 0 -1  
8 0 -4
```

对于实矩阵，转置运算对 a_{ij} 和 a_{ji} 进行交换。对于复矩阵，还要考虑是否用数组中复数项的复共轭来形成复共轭转置。MATLAB 使用撇号运算符 (`'`) 执行复共轭转置，使用点撇号运算符 (`.'`) 执行无共轭的转置。对于包含所有实数元素的矩阵，这两个运算符返回相同结果。

示例矩阵 $A = \text{pascal}(3)$ 是对称的，因此 A' 等于 A 。然而， $B = \text{magic}(3)$ 不是对称的，因此 B' 的元素是 B 的元素沿主对角线反转之后的结果：

```
B = magic(3)
```

```
B =
```

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

```
X = B'
```

```
X =
```

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

对于向量，转置会将行向量变为列向量（反之亦然）：

```
x = v'
```

```
x =
```

```
2
0
-1
```

如果 x 和 y 均为实数列向量，则乘积 $x*y$ 不确定，但以下两个乘积

```
x'*y
```

和

```
y'*x
```

产生相同的标量结果。此参数使用很频繁，它有三个不同的名称内积、标量积或点积。甚至还有一个专门的点积函数，称为 `dot`。

对于复数向量或矩阵 z ，参量 z' 不仅可转置该向量或矩阵，而且可将每个复数元素转换为其复共轭数。也就是说，每个复数元素的虚部的正负号将会发生更改。以如下复矩阵为例：

```
z = [1+2i 7-3i 3+4i; 6-2i 9i 4+7i]
```

```
z =
```

```
1.0000 + 2.0000i 7.0000 - 3.0000i 3.0000 + 4.0000i
6.0000 - 2.0000i 0.0000 + 9.0000i 4.0000 + 7.0000i
```

z 的复共轭转置为：

```
z'
```

```
ans =
```

```
1.0000 - 2.0000i 6.0000 + 2.0000i
7.0000 + 3.0000i 0.0000 - 9.0000i
3.0000 - 4.0000i 4.0000 - 7.0000i
```

非共轭复数转置（其中每个元素的复数部分保留其符号）表示为 $z.'$:

```
z.'
```

```
ans =
```

```
1.0000 + 2.0000i 6.0000 - 2.0000i
7.0000 - 3.0000i 0.0000 + 9.0000i
3.0000 + 4.0000i 4.0000 + 7.0000i
```

对于复数向量，两个标量积 x^*y 和 y^*x 互为复共轭数，而复数向量与其自身的标量积 x^*x 为实数。

矩阵乘法

矩阵乘法是以这样一种方式定义的：反映底层线性变换的构成，并允许紧凑表示联立线性方程组。如果 A 的列维度等于 B 的行维度，或者其中一个矩阵为标量，则可定义矩阵乘积 $C = AB$ 。如果 A 为 $m \times p$ 且 B 为 $p \times n$ ，则二者的乘积 C 为 $m \times n$ 。该乘积实际上可以使用 MATLAB for 循环、colon 表示法和向量点积进行定义：

```
A = pascal(3);
B = magic(3);
m = 3;
n = 3;
for i = 1:m
    for j = 1:n
        C(i,j) = A(i,:)*B(:,j);
    end
end
```

MATLAB 使用星号表示矩阵乘法，如 $C = A*B$ 中所示。矩阵乘法不适用交换律；即 $A*B$ 通常不等于 $B*A$ ：

```
X = A*B
```

```
X =
 15   15   15
 26   38   26
 41   70   39
```

```
Y = B*A
```

```
Y =
 15   28   47
 15   34   60
 15   28   43
```

矩阵可以在右侧乘以列向量，在左侧乘以行向量：

```
u = [3; 1; 4];
x = A*u
```

```
x =
```

17
30

```
v = [2 0 -1];  
y = v*B
```

y =

12 -7 10

矩形矩阵乘法必须满足维度兼容性条件：由于 A 是 3×3 矩阵，C 是 3×2 矩阵，因此可将二者相乘得到 3×2 结果（共同的内部维度会消去）：

X = A*C

X =

24 17
47 42
79 77

但是，乘法不能以相反的顺序执行：

Y = C*A

```
Error using *
Incorrect dimensions for matrix multiplication. Check that the number of columns
in the first matrix matches the number of rows in the second matrix. To perform
elementwise multiplication, use '.*'.
```

您可以将任何内容与标量相乘：

```
s = 10;
w = s*y
```

w =

120 -70 100

当您将数组与标量相乘时，标量将隐式扩展为与另一输入相同的大小。这通常称为标量扩展。

单位矩阵

普遍接受的数学表示法使用大写字母 I 来表示单位矩阵，即主对角线元素为 1 且其他位置元素为 0 的各种大小的矩阵。这些矩阵具有以下属性：无论维度是否兼容，AI = A 和 IA = A。

原始版本的 MATLAB 不能将 I 用于此用途，因为它不会区分大小字母和小写字母，并且 i 已用作下标和复数单位。因此，引入了英语双关语。函数

`eye(m,n)`

返回 $m \times n$ 矩形单位矩阵，`eye(n)` 返回 $n \times n$ 单位方阵。

矩阵求逆

如果矩阵 A 为非奇异方阵（非零行列式），则方程 $AX = I$ 和 $XA = I$ 具有相同的解 X。此解称为 A 的逆矩阵，表示为 A^{-1} 。`inv` 函数和表达式 A^{-1} 均可对矩阵求逆。

`A = pascal(3)`

```
A =
1   1   1
1   2   3
1   3   6

X = inv(A)

X =

3.0000  -3.0000  1.0000
-3.0000  5.0000  -2.0000
1.0000  -2.0000  1.0000
```

```
A*X

ans =

1.0000      0      0
0.0000  1.0000  -0.0000
-0.0000  0.0000  1.0000
```

通过 `det` 计算的行列式表示由矩阵描述的线性变换的缩放因子。当行列式正好为零时，矩阵为奇异矩阵，因此不存在逆矩阵。

```
d = det(A)
```

```
d =
```

```
1
```

有些矩阵接近奇异矩阵，虽然存在逆矩阵，但计算容易出现数值误差。`cond` 函数计算逆运算的条件数，它指示矩阵求逆结果的精度。条件数的范围是从 1（数值稳定的矩阵）到 Inf（奇异矩阵）。

```
c = cond(A)
```

```
c =
```

```
61.9839
```

很少需要为某个矩阵构造显式逆矩阵。求解线性方程组 $Ax = b$ 时，常常会误用 `inv`。从执行时间和数值精度方面而言，求解此方程的最佳方法是使用矩阵反斜杠运算符，即 $x = A \backslash b$ 。有关详细信息，请参阅 `mldivide`。

Kronecker 张量积

两个矩阵的 Kronecker 乘积 `kron(X,Y)` 为 X 的元素与 Y 的元素的所有可能乘积构成的较大矩阵。如果 X 为 $m \times n$ 且 Y 为 $p \times q$ ，则 `kron(X,Y)` 为 $mp \times nq$ 。元素以特定方式排列，呈现 X 的每个元素分别与整个矩阵 Y 相乘的结果。

```
[X(1,1)*Y  X(1,2)*Y  ...  X(1,n)*Y
 ...
X(m,1)*Y  X(m,2)*Y  ...  X(m,n)*Y]
```

Kronecker 乘积通常与元素为 0 和 1 的矩阵配合使用，以构建小型矩阵的重复副本。例如，如果 X 为 2×2 矩阵

```
X = [1  2
      3  4]
```

且 $I = \text{eye}(2,2)$ 为 2×2 单位矩阵，则：

```
kron(X,I)
```

```
ans =
```

```
1 0 2 0
0 1 0 2
3 0 4 0
0 3 0 4
```

并且

```
kron(I,X)
```

```
ans =
```

```
1 2 0 0
3 4 0 0
0 0 1 2
0 0 3 4
```

除了 **kron** 之外，对复制数组有用的其他函数还有 **repmat**、**repelem** 和 **blkdiag**。

向量范数和矩阵范数

向量 x 的 p -范数，

$$\|x\|_p = \left(\sum |x_i|^p \right)^{1/p},$$

使用 **norm(x,p)** 进行计算。此运算是为 $p > 1$ 的任意值定义的，但最常见的 p 值为 1、2 和 ∞ 。默认值为 $p = 2$ ，这与欧几里得长度或向量幅值对应：

```
v = [2 0 -1];
[norm(v,1) norm(v) norm(v,inf)]
```

```
ans =
```

```
3.0000 2.2361 2.0000
```

矩阵 A 的 p -范数，

$$\|A\|_p = \max_x \frac{\|Ax\|_p}{\|x\|_p},$$

可以针对 $p = 1$ 、 2 和 ∞ 通过 **norm(A,p)** 进行计算。同样，默认值也为 $p = 2$ ：

```
A = pascal(3);
[norm(A,1) norm(A) norm(A,inf)]
```

```
ans =
```

```
10.0000 7.8730 10.0000
```

如果要计算矩阵的每行或每列的范数，可以使用 **vecnorm**：

```
vecnorm(A)
```

```
ans =  
1.7321 3.7417 6.7823
```

使用线性代数方程函数的多线程计算

对于许多线性代数函数和按元素的数值函数，MATLAB 软件支持多线程计算。这些函数将自动在多个线程上执行。要使函数或表达式在多个 CPU 上更快地执行，必须满足许多条件：

- 1 函数执行的运算可轻松划分为并发执行的多个部分。这些部分必须能够在进程之间几乎不通信的情况下执行。它们应需要很少的序列运算。
- 2 数据大小足以使并发执行的任何优势在重要性方面超过对数据分区和管理各个执行线程所需的时间。例如，仅当数组包含数千个或以上的元素时，大多数函数才会加速。
- 3 运算未与内存绑定；处理时间不受内存访问时间控制。一般而言，复杂函数比简单函数速度更快。

对于大型双精度数组（约 10,000 个元素），矩阵乘法 ($X \cdot Y$) 和矩阵乘幂 (X^p) 运算符会大幅增加速度。矩阵分析函数 `det`、`rcond`、`hess` 和 `expm` 也会对大型双精度数组大幅增加速度。

线性方程组

本节内容

- “计算注意事项”（第 2-10 页）
- “通解”（第 2-11 页）
- “方阵方程组”（第 2-11 页）
- “超定方程组”（第 2-13 页）
- “欠定方程组”（第 2-15 页）
- “多右端线性方程组的求解”（第 2-16 页）
- “迭代法”（第 2-17 页）
- “多线程计算”（第 2-17 页）

计算注意事项

进行科学计算时，最重要的一个问题是对联立线性方程组求解。

在矩阵表示法中，常见问题采用以下形式：给定两个矩阵 A 和 b ，是否存在一个唯一矩阵 x 使 $Ax = b$ 或 $xA = b$ ？

考虑 1×1 示例具有指导意义。例如，方程

$$7x = 21$$

是否具有唯一解？

答案当然是肯定的。方程有唯一解 $x = 3$ 。通过除法很容易求得该解：

$$x = 21/7 = 3。$$

该解通常不是通过计算 7 的倒数求得的，即先计算 $7^{-1} = 0.142857\dots$ ，然后将 7^{-1} 乘以 21。这将需要更多的工作，而且如果 7^{-1} 以有限位数表示时，准确性会较低。类似注意事项也适用于多个未知数的线性方程组；MATLAB 在解此类方程时不会计算矩阵的逆。

尽管这不是标准的数学表示法，但 MATLAB 使用标量示例中常见的除法术语来描述常规联立方程组的解。斜杠 / 和反斜杠 \ 这两个除号分别对应 MATLAB 函数 `mrdivide` 和 `mldivide`。两种运算符分别用于未知矩阵出现在系数矩阵左侧或右侧的情况：

- | | |
|-----------|---|
| $x = b/A$ | 表示使用 <code>mrdivide</code> 获得的矩阵方程 $xA = b$ 的解。 |
| $x = A\b$ | 表示使用 <code>mldivide</code> 获得的矩阵方程 $Ax = b$ 的解。 |

考虑将方程 $Ax = b$ 或 $xA = b$ 的两端“除以” A 。系数矩阵 A 始终位于“分母”中。

$x = A\b$ 的维度兼容性条件要求两个矩阵 A 和 b 的行数相同。这样，解 x 的列数便与 b 的列数相同，并且其行维度等于 A 的列维度。对于 $x = b/A$ ，行和列的角色将会互换。

实际上， $Ax=b$ 形式的线性方程组比 $xA=b$ 形式的线性方程组更常见。因此，反斜杠的使用频率要远高于斜杠的使用频率。本节其余部分将重点介绍反斜杠运算符；斜杠运算符的对应属性可以从以下恒等式推知：

$$(b/A)' = (A'\backslash b').$$

系数矩阵 A 不需要是方阵。如果 A 的大小为 $m \times n$, 则有三种情况:

$m = n$	方阵方程组。求精确解。
$m > n$	超定方程组, 即方程个数多于未知数个数。求最小二乘解。
$m < n$	欠定方程组, 即方程个数少于未知数个数。使用最多 m 个非零分量求基本解。

mldivide 算法

mldivide 运算符使用不同的求解器来处理不同类型的系数矩阵。通过检查系数矩阵自动诊断各种情况。有关详细信息, 请参阅 **mldivide** 参考页的“算法”小节。

通解

线性方程组 $Ax = b$ 的通解描述了所有可能的解。可以通过以下方法求通解:

- 1 求对应的齐次方程组 $Ax = 0$ 的解。使用 **null** 命令通过键入 **null(A)** 来执行此操作。这会将解空间的基向量恢复为 $Ax = 0$ 。任何解都是基向量的线性组合。
- 2 求非齐次方程组 $Ax = b$ 的特定解。

然后, 可将 $Ax = b$ 的任何解写成第 2 步中的 $Ax = b$ 的特定解加上第 1 步中的基向量的线性组合之和。

本节其余部分将介绍如何使用 MATLAB 求 $Ax = b$ 的特定解, 如第 2 步中所述。

方阵方程组

最常见的涉及一个方阵系数矩阵 A 和一个右侧单列向量 b 。

非奇异系数矩阵

如果矩阵 A 是非奇异矩阵, 则解 $x = A \setminus b$ 的大小与 b 的大小相同。例如:

```
A = pascal(3);
u = [3; 1; 4];
x = A \ u

x =
    10
   -12
     5
```

可以确认 $A * x$ 恰好等于 u 。

如果 A 和 b 为方阵并且大小相同, 则 $x = A \setminus b$ 也具有相同大小:

```
b = magic(3);
X = A \ b

X =
    19   -3   -1
   -17    4   13
     6    0   -6
```

可以确认 $A * x$ 恰好等于 b 。

以上两个示例具有确切的整数解。这是因为系数矩阵选为 `pascal(3)`，这是满秩矩阵（非奇异的）。

奇异系数矩阵

如果方阵 A 不包含线性无关的列，则该矩阵为奇异矩阵。如果 A 为奇异矩阵，则 $Ax = b$ 的解将不存在或不唯一。如果 A 接近奇异或检测到完全奇异性，则反斜杠运算符 `A\b` 会发出警告。

如果 A 为奇异矩阵并且 $Ax = b$ 具有解，可以通过键入以下内容求不是唯一的特定解

```
P = pinv(A)*b
```

`pinv(A)` 是 A 的伪逆。如果 $Ax = b$ 没有精确解，则 `pinv(A)` 将返回最小二乘解。

例如：

```
A = [ 1   3   7
      -1  4   4
       1  10  18 ]
```

为奇异矩阵，可以通过键入以下内容进行验证：

```
rank(A)
```

```
ans =
```

```
2
```

由于 A 不是满秩，它有一些等于零的奇异值。

精确解。对于 `b = [5;2;12]`，方程 $Ax = b$ 具有精确解，给定

```
pinv(A)*b
```

```
ans =
0.3850
-0.1103
0.7066
```

通过键入以下内容验证 `pinv(A)*b` 是否为精确解

```
A*pinv(A)*b
```

```
ans =
5.0000
2.0000
12.0000
```

最小二乘解。但是，如果 `b = [3;6;0]`，则 $Ax = b$ 没有精确解。在这种情况下，`pinv(A)*b` 会返回最小二乘解。键入

```
A*pinv(A)*b
```

```
ans =
-1.0000
4.0000
2.0000
```

则不会返回原始向量 `b`。

通过得到增广矩阵 [A b] 的简化行阶梯形式，可以确定 $Ax = b$ 是否具有精确解。为此，对于此示例请输入

```
rref([A b])
ans =
1.0000   0   2.2857   0
0   1.0000   1.5714   0
0   0       0   1.0000
```

由于最下面一行全部为零（最后一项除外），因此该方程无解。在这种情况下，**pinv(A)** 会返回最小二乘解。

超定方程组

此示例说明在对试验数据的各种曲线拟合中通常会如何遇到超定方程组。

在多个不同的时间值 t 对数量 y 进行测量以生成以下观测值。可以使用以下语句输入数据并在表中查看该数据。

```
t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';
B = table(t,y)
```

```
B=6×2 table
t      y
_____

```

0	0.82
0.3	0.72
0.8	0.63
1.1	0.6
1.6	0.55
2.3	0.5

尝试使用指数衰减函数对数据进行建模

$$y(t) = c_1 + c_2 e^{-t}.$$

上一方程表明，向量 y 应由两个其他向量的线性组合来逼近。一个是元素全部为 1 的常向量，另一个是带有分量 $\exp(-t)$ 的向量。未知系数 c_1 和 c_2 可以通过执行最小二乘拟合来计算，这样会最大限度地减小数据与模型偏差的平方和。在两个未知系数的情况下有六个方程，用 6×2 矩阵表示。

```
E = [ones(size(t)) exp(-t)]
```

```
E = 6×2
```

1.0000	1.0000
1.0000	0.7408
1.0000	0.4493
1.0000	0.3329
1.0000	0.2019
1.0000	0.1003

使用反斜杠运算符获取最小二乘解。

```
c = E\y
```

```
c = 2×1
```

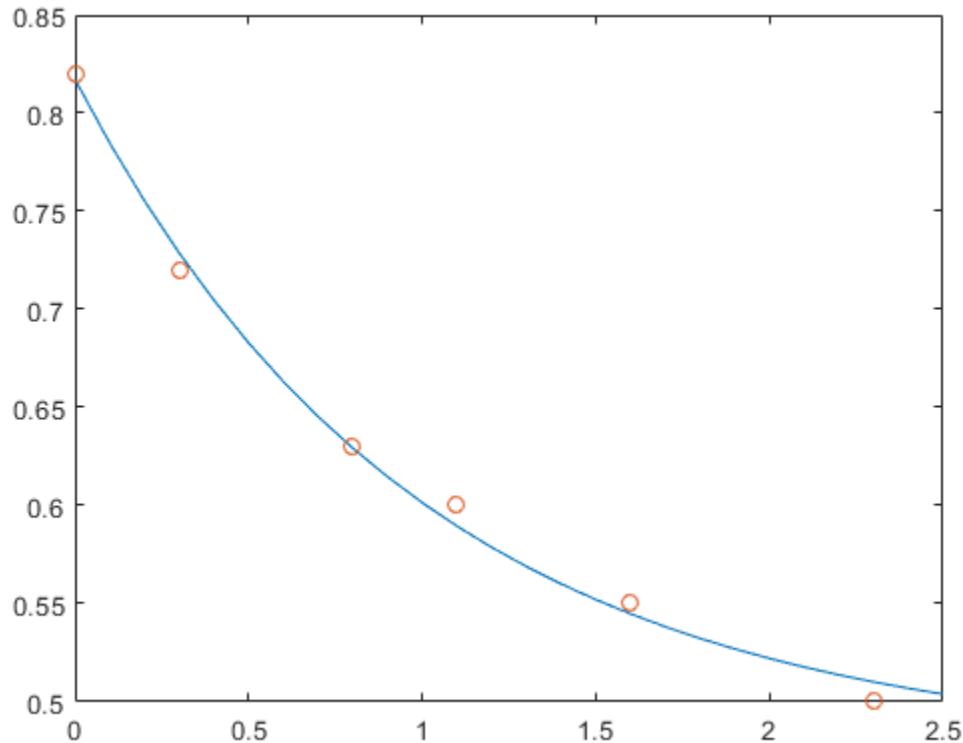
```
0.4760  
0.3413
```

也就是说，对数据的最小二乘拟合为

$$y(t) = 0.4760 + 0.3413e^{-t}.$$

以下语句按固定间隔的 t 增量为模型求值，然后与原始数据一同绘制结果：

```
T = (0:0.1:2.5)';  
Y = [ones(size(T)) exp(-T)]*c;  
plot(T,Y,'-',t,y,'o')
```



E^*c 与 y 不完全相等，但差值可能远小于原始数据中的测量误差。

如果矩形矩阵 A 没有线性无关的列，则该矩阵秩亏。如果 A 秩亏，则 $AX = B$ 的最小二乘解不唯一。如果 A 秩亏，则 $A \setminus B$ 会发出警告，并生成一个最小二乘解。您可以使用 `lsqminnorm` 求在所有解中具有最小范数的解 X 。

欠定方程组

本例演示了欠定方程组的解不唯一的情况。欠定线性方程组包含的未知数比方程多。MATLAB 矩阵左除运算求基本最小二乘解，对于 $m \times n$ 系数矩阵，它最多有 m 个非零分量。

以下是一个简单的随机示例：

```
R = [6 8 7 3; 3 5 4 1]
rng(0);
b = randi(8,2,1)
```

R =

6	8	7	3
3	5	4	1

b =

7
8

线性方程组 $Rp = b$ 有两个方程，四个未知数。由于系数矩阵包含较小的整数，因此适合使用 **format rat** 命令以有理格式显示解。通过以下命令可获取特定解

```
format rat
p = R\b
```

p =

0
17/7
0
-29/7

其中一个非零分量为 **p(2)**，因为 $R(:,2)$ 是具有最大范数的 R 的列。另一个非零分量为 **p(4)**，因为 $R(:,4)$ 在消除 $R(:,2)$ 后起控制作用。

欠定方程组的完全通解可以通过 **p** 加上任意零空间向量线性组合来表示，可以使用 **null** 函数（使用请求有理基的选项）计算该空间向量。

```
Z = null(R,'r')
```

Z =

-1/2	-7/6
-1/2	1/2
1	0
0	1

可以确认 $R*Z$ 为零，并且残差 $R*x - b$ 远远小于任一向量 x （其中

$x = p + Z*q$

由于 Z 的列是零空间向量，因此 $Z*q$ 是以下向量的线性组合：

$$Zq = (\vec{x}_1 \ \vec{x}_2) \begin{pmatrix} u \\ w \end{pmatrix} = u\vec{x}_1 + w\vec{x}_2 .$$

为了说明这一点，选择任意 q 并构造 x 。

```
q = [-2; 1];
x = p + Z*q;
```

计算残差的范数。

```
format short
norm(R*x - b)
```

```
ans =
```

```
2.6645e-15
```

如果有无限多个解，则最小范数解具有特别意义。您可以使用 **lsqminnorm** 计算最小范数最小二乘解。该解具有 **norm(p)** 的最小可能值。

```
p = lsqminnorm(R,b)
```

```
p =
```

```
-207/137
365/137
79/137
-424/137
```

多右端线性方程组的求解

某些问题涉及求解具有相同系数矩阵 A 但具有不同右端 b 的线性方程组。如果可以同时使用不同的 b 值，则可以将 b 构造为多列矩阵，并使用单个反斜杠命令求解所有方程组： $X = A \backslash [b_1 \ b_2 \ b_3 \ ...]$ 。

但是，有时不同的 b 值并非全部同时可用，也就是说，您需要连续求解若干方程组。如果使用斜杠 (/) 或反斜杠 (\) 求解其中一个方程组，则该运算符会对系数矩阵 A 进行分解，并使用此矩阵分解来求解。然而，随后每次使用不同的 b 求解类似方程组时，运算符都会对 A 进行同样的分解，而这是一次冗余计算。

此问题的求解是预先计算 A 的分解，然后重新使用因子对 b 的不同值求解。但是，实际上，以这种方式预先计算分解可能很困难，因为需要知道要计算的分解 (LU、LDL、Cholesky 等) 以及如何乘以因子才能对问题求解。例如，使用 LU 分解，您需要求解两个线性方程组才能求解原始方程组 $Ax = b$ ：

```
[L,U] = lu(A);
x = U \ (L \ b);
```

对于具有若干连续右端的线性方程组，建议使用 **decomposition** 对象求解。借助这些对象，您可利用预先计算矩阵分解带来的性能优势，而不必了解如何使用矩阵因子。您可以将先前的 LU 分解替换为：

```
dA = decomposition(A,'lu');
x = dA\ b;
```

如果您不确定要使用哪种分解，**decomposition(A)** 会根据 A 的属性选择正确的类型，类似于反斜杠的功能。

以下简单测试验证了此方法可能带来的性能优势。该测试分别使用反斜杠 (\) 和 **decomposition** 对同一稀疏线性方程组求解 100 次。

```
n = 1e3;
A = sprand(n,n,0.2) + speye(n);
b = ones(n,1);
```

```
% Backslash solution
tic
for k = 1:100
    x = A\b;
end
toc
```

Elapsed time is 9.006156 seconds.

```
% decomposition solution
tic
dA = decomposition(A);
for k = 1:100
    x = dA\b;
end
toc
```

Elapsed time is 0.374347 seconds.

对于这个问题，**decomposition** 求解比单独使用反斜杠要快得多，而语法仍然很简单。

迭代法

如果系数矩阵 A 很大并且是稀疏矩阵，分解方法一般情况下将不会有效。迭代方法可生成一系列近似解。MATLAB 提供了多个迭代方法来处理大型的稀疏输入矩阵。

函数	说明
pcg	预条件共轭梯度法。此方法适用于 Hermitian 正定系数矩阵 A。
bicg	双共轭梯度法
bicgstab	双共轭梯度稳定法
bicgstabl	双共轭梯度稳定法(l)
cgs	共轭梯度二乘法
gmres	广义最小残差法
lsqr	LSQR 方法
minres	最小残差法。此方法适用于 Hermitian 系数矩阵 A。
qmr	拟最小残差法
symmlq	对称的 LQ 方法
tfqmr	无转置 QMR 方法

多线程计算

对于许多线性代数函数和按元素的数值函数，MATLAB 软件支持多线程计算。这些函数将自动在多个线程上执行。要使函数或表达式在多个 CPU 上更快地执行，必须满足许多条件：

- 1 函数执行的运算可轻松划分为并发执行的多个部分。这些部分必须能够在进程之间几乎不通信的情况下执行。它们应需要很少的序列运算。
- 2 数据大小足以使并发执行的任何优势在重要性方面超过对数据分区和管理各个执行线程所需的时间。例如，仅当数组包含数千个或以上的元素时，大多数函数才会加速。

3 运算未与内存绑定；处理时间不受内存访问时间控制。一般而言，复杂函数比简单函数速度更快。

如果启用多线程，`inv`、`lscov`、`linsolve` 和 `mldivide` 将会对大型双精度数组（约 10,000 个元素或更多）大幅增加速度。

分解

本节内容

- “简介” (第 2-19 页)
- “Cholesky 分解” (第 2-19 页)
- “LU 分解” (第 2-20 页)
- “QR 分解” (第 2-21 页)
- “对分解使用多线程计算” (第 2-23 页)

简介

本节中讨论的所有三种矩阵分解利用了三角形矩阵，其中对角线上的所有元素都为零。涉及三角矩阵的线性方程组可以使用前代或回代方法轻松快捷地求解。

Cholesky 分解

Cholesky 分解将对称矩阵表示为三角矩阵与其转置的积

$$A = R'R,$$

其中，R 是上三角矩阵。

并非所有对称矩阵都可以通过这种方式进行分解；采用此类分解的矩阵被视为正定矩阵。这表明，A 的所有对角线元素都是正数，并且非对角线元素“不太大”。帕斯卡矩阵提供了有趣的示例。在本章中，示例矩阵 A 为 3×3 帕斯卡矩阵。暂时转换为 6×6 ：

$$A = \text{pascal}(6)$$

A =

$$\begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 3 & 6 & 10 & 15 & 21 \\ 1 & 4 & 10 & 20 & 35 & 56 \\ 1 & 5 & 15 & 35 & 70 & 126 \\ 1 & 6 & 21 & 56 & 126 & 252 \end{matrix}$$

A 的元素为二项式系数。每个元素都是其北向和西向邻点之和。Cholesky 分解为

$$R = \text{chol}(A)$$

R =

$$\begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 1 & 3 & 6 & 10 \\ 0 & 0 & 0 & 1 & 4 & 10 \\ 0 & 0 & 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

这些元素同样为二项式系数。 R'^*R 等于 A 的情况说明了涉及二项式系数的积之和的单位矩阵。

注意 Cholesky 分解也适用于复矩阵。采用 Cholesky 分解的任何复矩阵都满足

$$A' = A$$

, 并且被视为 Hermitian 正定矩阵。

通过 Cholesky 分解, 可以将线性方程组

$$Ax = b$$

替换为

$$R'R x = b.$$

由于反斜杠运算符能识别三角形方程组, 因此这可以在 MATLAB 环境中通过以下表达式快速进行求解

$$x = R \backslash (R' \backslash b)$$

如果 A 为 $n \times n$, 则 $\text{chol}(A)$ 的计算复杂度为 $O(n^3)$, 但后续的反斜杠解的复杂度仅为 $O(n^2)$ 。

LU 分解

LU 分解 (或高斯消去法) 将任何方阵 A 都表示为下三角矩阵和上三角矩阵的置换之积

$$A = LU,$$

其中, L 是对角线元素为 1 的下三角形矩阵的置换, U 是上三角形矩阵。

出于理论和计算原因, 必须进行置换。矩阵

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

在不交换其两行的情况下不能表示为三角矩阵的积。尽管矩阵

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 0 \end{bmatrix}$$

可以表示为三角矩阵之积, 但当 ε 很小时, 因子中的元素也会很大并且会增大误差, 因此即使置换并非完全必要, 它们也是所希望的。部分主元消元法可确保 L 的元素的模以 1 为限, 并且 U 的元素并不大于 A 的元素。

例如:

$$[L, U] = lu(B)$$

$$L =$$

$$\begin{matrix} 1.0000 & 0 & 0 \\ 0.3750 & 0.5441 & 1.0000 \\ 0.5000 & 1.0000 & 0 \end{matrix}$$

$$U =$$

$$\begin{matrix} 8.0000 & 1.0000 & 6.0000 \\ 0 & 8.5000 & -1.0000 \\ 0 & 0 & 5.2941 \end{matrix}$$

通过对 A 执行 LU 分解, 可以

$$A^*x = b$$

使用以下表达式快速对线性方程组求解

$$x = U \setminus (L \setminus b)$$

行列式和逆矩阵是通过 LU 分解使用以下表达式进行计算的

$$\det(A) = \det(L) * \det(U)$$

和

$$\text{inv}(A) = \text{inv}(U) * \text{inv}(L)$$

也可以使用 $\det(A) = \text{prod}(\text{diag}(U))$ 计算行列式，但行列式的符号可能会相反。

QR 分解

正交矩阵或包含正交列的矩阵为实矩阵，其列全部具有单位长度并且相互垂直。如果 Q 为正交矩阵，则

$$Q'Q = I.$$

最简单的正交矩阵为二维坐标旋转：

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}.$$

对于复矩阵，对应的术语为单位。由于正交矩阵和单位矩阵会保留长度、保留角度并且不会增大误差，因此适用于数值计算。

正交或 QR 分解将任何矩形矩阵表示为正交或酉矩阵与上三角矩阵的积。此外，也可能涉及列置换。

$$A = QR$$

或

$$AP = QR,$$

其中，Q 为正交或单位矩阵，R 为上三角矩阵，P 为置换向量。

QR 分解有四种变化形式 - 完全大小或合适大小，以及使用列置换或不使用列置换。

超定线性方程组涉及行数超过列数的矩形矩阵，也即 $m > n$ 并且 $m > n$ 。完全大小的 QR 分解可生成一个方阵 ($m \times m$ 正交矩阵 Q) 和一个矩形 $m \times n$ 上三角矩阵 R:

```
C=gallery('uniformdata',[5 4], 0);
[Q,R] = qr(C)
```

$$Q =$$

0.6191	0.1406	-0.1899	-0.5058	0.5522
0.1506	0.4084	0.5034	0.5974	0.4475
0.3954	-0.5564	0.6869	-0.1478	-0.2008
0.3167	0.6676	0.1351	-0.1729	-0.6370
0.5808	-0.2410	-0.4695	0.5792	-0.2207

$R =$

1.5346	1.0663	1.2010	1.4036
0	0.7245	0.3474	-0.0126
0	0	0.9320	0.6596
0	0	0	0.6648
0	0	0	0

在许多情况下， Q 的最后 $m - n$ 列是不需要的，因为这些列会与 R 底部的零相乘。因此，精简 QR 分解可生成一个矩形矩阵（具有正交列的 $m \times n Q$ ）以及一个方阵 $n \times n$ 上三角形矩阵 R 。对于 5×4 示例，不会节省太多内存，但是对于更大的大量矩形矩阵，在时间和内存方面的节省可能会很重要：

 $[Q, R] = qr(C, 0)$ $Q =$

0.6191	0.1406	-0.1899	-0.5058
0.1506	0.4084	0.5034	0.5974
0.3954	-0.5564	0.6869	-0.1478
0.3167	0.6676	0.1351	-0.1729
0.5808	-0.2410	-0.4695	0.5792

 $R =$

1.5346	1.0663	1.2010	1.4036
0	0.7245	0.3474	-0.0126
0	0	0.9320	0.6596
0	0	0	0.6648

与 LU 分解相比，QR 分解不需要进行任何消元或置换。但是，可选的列置换（因存在第三个输出参数而触发）对检测奇异性和秩亏是很有帮助的。在分解的每一步，未分解的剩余矩阵的列（范数最大）将用作该步骤的基础。这可以确保 R 的对角线元素以降序排列，并且各列之间的任何线性相关性肯定能够通过检查这些元素来显示。对于此处提供的小示例， C 的第二列的范数大于第一列的范数，因此这两列被交换：

 $[Q, R, P] = qr(C)$ $Q =$

-0.3522	0.8398	-0.4131
-0.7044	-0.5285	-0.4739
-0.6163	0.1241	0.7777

 $R =$

-11.3578	-8.2762
0	7.2460
0	0

 $P =$

0	1
1	0

组合了合适大小和列置换后，第三个输出参数为置换向量而不是置换矩阵：

 $[Q, R, p] = qr(C, 0)$ $Q =$

-0.3522	0.8398
-0.7044	-0.5285
-0.6163	0.1241

```
R =
-11.3578 -8.2762
  0  7.2460
```

```
p =
 2  1
```

QR 分解可将超定线性方程组转换为等效的三角形方程组。表达式

```
norm(A*x - b)
```

等于

```
norm(Q*R*x - b)
```

与正交矩阵相乘可保留欧几里德范数，因此该表达式也等于

```
norm(R*x - y)
```

其中 $y = Q^T * b$ 。由于 R 的最后 $m-n$ 行为零，因此该表达式将分为两部分：

```
norm(R(1:n,1:n)*x - y(1:n))
```

并且

```
norm(y(n+1:m))
```

如果 A 具有满秩，则可以对 x 求解，使这些表达式中的第一个表达式为零。然后，第二个表达式便可以提供残差范数。如果 A 没有满秩，则可以通过 R 的三角形结构体对最小二乘问题求基本解。

对分解使用多线程计算

对于许多线性代数函数和按元素的数值函数，MATLAB 软件支持多线程计算。这些函数将自动在多个线程上执行。要使函数或表达式在多个 CPU 上更快地执行，必须满足许多条件：

- 1 函数执行的运算可轻松划分为并发执行的多个部分。这些部分必须能够在进程之间几乎不通信的情况下执行。它们应需要很少的序列运算。
- 2 数据大小足以使并发执行的任何优势在重要性方面超过对数据分区和管理各个执行线程所需的时间。例如，仅当数组包含数千个或以上的元素时，大多数函数才会加速。
- 3 运算未与内存绑定；处理时间不受内存访问时间控制。一般而言，复杂函数比简单函数速度更快。

对于大型双精度数组（约 10,000 个元素），lu 和 qr 会大幅增加速度。

幂和指数

本主题说明如何使用各种方法计算矩阵幂和指数。

正整数幂

如果 A 为方阵并且 p 为正整数，则 A^p 实际上是将 A 乘以其自身 p-1 次。例如：

```
A = [1 1 1
      1 2 3
      1 3 6];
A^2
```

ans = 3×3

```
3   6   10
6   14  25
10  25  46
```

逆幂和分数幂

如果 A 为方阵并且是非奇异的，则 A^{-p} 实际上是将 $\text{inv}(A)$ 乘以其自身 p-1 次。

A^{-3}

ans = 3×3

```
145.0000 -207.0000  81.0000
-207.0000 298.0000 -117.0000
 81.0000 -117.0000  46.0000
```

MATLAB® 用相同的算法计算 $\text{inv}(A)$ 和 A^{-1} ，因此结果完全相同。如果矩阵接近奇异， $\text{inv}(A)$ 和 A^{-1} 都会产生警告。

`isequal(inv(A),A-1)`

```
ans = logical
  1
```

也允许分数幂，例如 $A^{(2/3)}$ 。使用小数幂的结果取决于矩阵特征值的分布。

$A^{(2/3)}$

ans = 3×3

```
0.8901  0.5882  0.3684
0.5882  1.2035  1.3799
 0.3684  1.3799  3.1167
```

逐元素幂

.^ 运算符计算逐元素幂。例如，要对矩阵中的每个元素求平方，可以使用 $A.^2$ 。

$A.^2$

```
ans = 3×3
```

1	1	1
1	4	9
1	9	36

平方根

使用 **sqrt** 函数可以方便地计算矩阵中每个元素的平方根。另一种方法是 **A.^^(1/2)**。

sqrt(A)

```
ans = 3×3
```

1.0000	1.0000	1.0000
1.0000	1.4142	1.7321
1.0000	1.7321	2.4495

对于其他根，您可以使用 **nthroot**。例如，计算 **A.^^(1/3)**。

nthroot(A,3)

```
ans = 3×3
```

1.0000	1.0000	1.0000
1.0000	1.2599	1.4422
1.0000	1.4422	1.8171

这些按元素计算的根不同于矩阵平方根，后者计算得到的是另一个矩阵 **B** 以满足 **A = BB**。函数 **sqrtm(A)** 采用更精确的算法计算 **A.^^(1/2)**。**sqrtm** 中的 **m** 将此函数与 **sqrt(A)** 区分开来，后者与 **A.^^(1/2)** 一样，以逐元素方式工作。

B = sqrtm(A)

```
B = 3×3
```

0.8775	0.4387	0.1937
0.4387	1.0099	0.8874
0.1937	0.8874	2.2749

B^2

```
ans = 3×3
```

1.0000	1.0000	1.0000
1.0000	2.0000	3.0000
1.0000	3.0000	6.0000

标量底

除了对矩阵求幂以外，您还可以以矩阵为次数对标量求幂。

2^A

```
ans = 3×3
10.4630 21.6602 38.5862
21.6602 53.2807 94.6010
38.5862 94.6010 173.7734
```

当您以矩阵为次数对标量求幂时，MATLAB 使用矩阵的特征值和特征向量来计算矩阵幂。如果 $[V,D] = \text{eig}(A)$ ，则 $2^A = V 2^D V^{-1}$ 。

```
[V,D] = eig(A);
V*2^D*V^(-1)
```

```
ans = 3×3
10.4630 21.6602 38.5862
21.6602 53.2807 94.6010
38.5862 94.6010 173.7734
```

矩阵指数

矩阵指数是以矩阵为次数对标量求幂的特殊情况。矩阵指数的底是欧拉数 $e = \exp(1)$ 。

```
e = exp(1);
e^A
```

```
ans = 3×3
103 ×
0.1008 0.2407 0.4368
0.2407 0.5867 1.0654
0.4368 1.0654 1.9418
```

expm 函数是计算矩阵指数的一种更方便的方法。

expm(A)

```
ans = 3×3
103 ×
0.1008 0.2407 0.4368
0.2407 0.5867 1.0654
0.4368 1.0654 1.9418
```

矩阵指数可以用多种方法来计算。有关详细信息，请参阅“矩阵指数”（第 2-35 页）。

处理较小的数字

对于非常小的值 x ，MATLAB 函数 **log1p** 和 **expm1** 可以精确计算 $\log(1 + x)$ 和 $e^x - 1$ 。例如，如果您尝试将小于计算机精度的一个数与 1 相加，则结果会舍入到 1。

log(1+eps/2)

ans = 0

但是，**log1p** 能够返回更准确的答案。

```
log1p(eps/2)
```

```
ans = 1.1102e-16
```

同样，对于 $e^x - 1$ ，如果 x 非常小，则会将它舍入为零。

```
exp(eps/2)-1
```

```
ans = 0
```

同样，`expm1` 能够返回更准确的答案。

```
expm1(eps/2)
```

```
ans = 1.1102e-16
```

特征值

本节内容

- “特征值的分解” (第 2-28 页)
- “多重特征值” (第 2-29 页)
- “Schur 分解” (第 2-29 页)

特征值的分解

方阵 A 的特征值和特征向量分别为满足以下条件的标量 λ 和非零向量 v

$$Av = \lambda v.$$

对于对角矩阵的对角线上的特征值 Λ 以及构成矩阵列的对应特征向量 V , 公式为

$$AV = V\Lambda.$$

如果 V 是非奇异的, 这将变为特征值分解。

$$A = V\Lambda V^{-1}.$$

微分方程 $dx/dt = Ax$ 的系数矩阵就是一个很好的示例:

$$A = \begin{pmatrix} 0 & -6 & -1 \\ 6 & 2 & -16 \\ -5 & 20 & -10 \end{pmatrix}$$

此方程的解用矩阵指数 $x(t) = e^{tA}x(0)$ 表示。语句

```
lambda = eig(A)
```

生成包含 A 的特征值的列向量。对于该矩阵, 这些特征值为复数:

$$\begin{aligned} \text{lambda} = \\ -3.0710 \\ -2.4645+17.6008i \\ -2.4645-17.6008i \end{aligned}$$

每个特征值的实部都为负数, 因此随着 t 的增加, $e^{\lambda t}$ 将会接近零。两个特征值 $\pm\omega$ 的非零虚部为微分方程的解提供了振动分量 $\sin(\omega t)$ 。

使用这两个输出参数, `eig` 便可以计算特征向量并将特征值存储在对角矩阵中:

```
[V,D] = eig(A)
```

$$\begin{aligned} V = \\ -0.8326 & 0.2003 - 0.1394i & 0.2003 + 0.1394i \\ -0.3553 & -0.2110 - 0.6447i & -0.2110 + 0.6447i \\ -0.4248 & -0.6930 & -0.6930 \end{aligned}$$

$$\begin{aligned} D = \\ -3.0710 & & 0 & & 0 \end{aligned}$$

```

0      -2.4645+17.6008i      0
0          0      -2.4645-17.6008i

```

第一个特征向量为实数，另外两个向量互为共轭复数。所有三个向量都归一化为具有等于 1 的欧几里德长度 $\text{norm}(v,2)$ 。

矩阵 $V^*D^*\text{inv}(V)$ (可更简洁地写为 V^*D/V) 位于 A 的舍入误差界限内。 $\text{inv}(V)^*A^*V$ 或 $V\backslash A^*V$ 都在 D 的舍入误差界限内。

多重特征值

某些矩阵没有特征向量分解。这些矩阵是不可对角化的。例如：

```

A = [ 1   -2   1
      0    1   4
      0    0   3 ]

```

对此矩阵

```
[V,D] = eig(A)
```

生成

$V =$

```

1.0000  1.0000 -0.5571
0    0.0000  0.7428
0    0    0.3714

```

$D =$

```

1   0   0
0   1   0
0   0   3

```

$\lambda = 1$ 时有一个双精度特征值。 V 的第一列和第二列相同。对此矩阵，并不存在一组完整的线性无关特征向量。

Schur 分解

许多高级矩阵计算不需要进行特征值分解。而是使用 Schur 分解。

$A = USU'$,

其中， U 是正交矩阵， S 是对角线上为 1×1 和 2×2 块的块上三角矩阵。特征值是通过 S 的对角元素和块显示的，而 U 的列提供了正交基，它的数值属性要远远优于一组特征向量。

例如，比较下面的亏损矩阵的特征值和 Schur 分解：

```

A = [ 6   12   19
      -9  -20  -33
      4    9   15 ];

```

```
[V,D] = eig(A)
```

$V =$

$$\begin{array}{cccc} -0.4741 + 0.0000i & -0.4082 - 0.0000i & -0.4082 + 0.0000i \\ 0.8127 + 0.0000i & 0.8165 + 0.0000i & 0.8165 + 0.0000i \\ -0.3386 + 0.0000i & -0.4082 + 0.0000i & -0.4082 - 0.0000i \end{array}$$
 $D =$

$$\begin{array}{ccc} -1.0000 + 0.0000i & 0.0000 + 0.0000i & 0.0000 + 0.0000i \\ 0.0000 + 0.0000i & 1.0000 + 0.0000i & 0.0000 + 0.0000i \\ 0.0000 + 0.0000i & 0.0000 + 0.0000i & 1.0000 - 0.0000i \end{array}$$
 $[U,S] = \text{schur}(A)$ $U =$

$$\begin{array}{ccc} -0.4741 & 0.6648 & 0.5774 \\ 0.8127 & 0.0782 & 0.5774 \\ -0.3386 & -0.7430 & 0.5774 \end{array}$$
 $S =$

$$\begin{array}{ccc} -1.0000 & 20.7846 & -44.6948 \\ 0 & 1.0000 & -0.6096 \\ 0 & 0.0000 & 1.0000 \end{array}$$

矩阵 A 为亏损矩阵，因为它不具备一组完整的线性无关特征向量（ V 的第二列和第三列相同）。由于 V 的列并非全部是线性无关的，因此它有一个很大的条件数，大约为 $1e8$ 。但 schur 可以计算 U 中的三个不同基向量。由于 U 是正交矩阵，因此 $\text{cond}(U) = 1$ 。

矩阵 S 的实数特征值作为对角线上的第一个条目，并通过右下方的 2×2 块表示重复的特征值。 2×2 块的特征值也是 A 的特征值：

 $\text{eig}(S(2:3,2:3))$ $ans =$

$$\begin{array}{c} 1.0000 + 0.0000i \\ 1.0000 - 0.0000i \end{array}$$

奇异值

矩形矩阵 A 的奇异值和对应的奇异向量分别为满足以下条件的标量 σ 以及一对向量 u 和 v

$$Av = \sigma u$$

$$A^H u = \sigma v,$$

其中 A^H 是 A 的 Hermitian 转置。奇异向量 u 和 v 通常缩放至范数为 1。此外，如果 u 和 v 均为 A 的奇异向量，则 $-u$ 和 $-v$ 也为 A 的奇异向量。

奇异值 σ 始终为非负实数，即使 A 为复数也是如此。对于对角矩阵 Σ 的对角线上的奇异值以及构成两个正交矩阵 U 和 V 的列的对应奇异向量，方程为

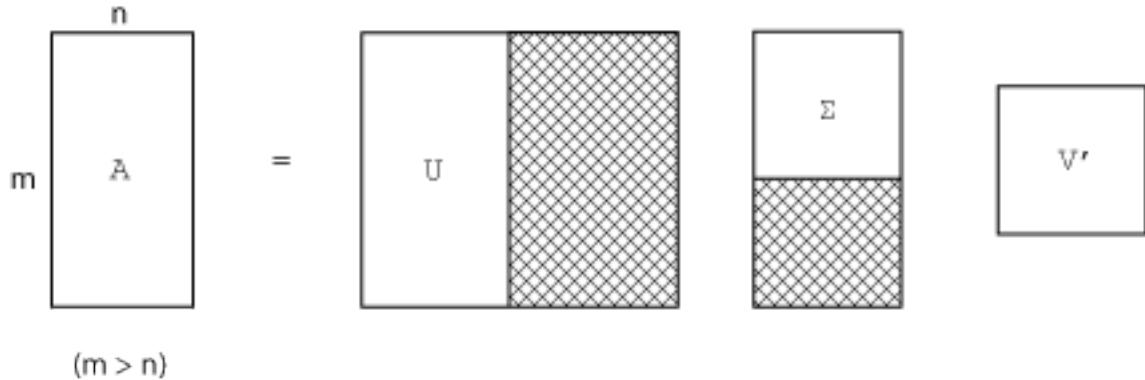
$$AV = U\Sigma$$

$$A^H U = V\Sigma.$$

由于 U 和 V 均为单位矩阵，因此将第一个方程的右侧乘以 V^H 会生成奇异值分解方程

$$A = U\Sigma V^H.$$

$m \times n$ 矩阵的完整奇异值分解涉及 $m \times m$ U 、 $m \times n$ Σ 以及 $n \times n$ V 。换句话说， U 和 V 均为方阵， Σ 与 A 的大小相同。如果 A 的行数远多于列数 ($m > n$)，则得到的 $m \times m$ 矩阵 U 为大型矩阵。但是， U 中的大多数列与 Σ 中的零相乘。在这种情况下，精简分解可通过生成一个 $m \times n$ U 、一个 $n \times n$ Σ 以及相同的 V 来同时节省时间和存储空间：



特征值分解是分析矩阵（当矩阵表示从向量空间到其自身的映射时）的合适工具，就像分析常微分方程一样。但是，奇异值分解是分析从一个向量空间到另一个向量空间（可能具有不同的维度）的映射的合适工具。大多数联立线性方程组都属于这第二类。

如果 A 是方形的对称正定矩阵，则其特征值分解和奇异值分解相同。但是，当 A 偏离对称性和正定性时，这两种分解之间的差异就会增加。特别是，实矩阵的奇异值分解始终为实数，但非对称实矩阵的特征值分解可能为复数。

对于示例矩阵

$$A = \begin{matrix} 9 & 4 \end{matrix}$$

```
6  8  
2  7
```

完整的奇异值分解为

```
[U,S,V] = svd(A)
```

```
U =
```

```
0.6105 -0.7174  0.3355  
0.6646  0.2336 -0.7098  
0.4308  0.6563  0.6194
```

```
S =
```

```
14.9359      0  
0   5.1883  
0   0
```

```
V =
```

```
0.6925 -0.7214  
0.7214  0.6925
```

可以验证 $U \cdot S \cdot V'$ 在舍入误差界限内是否等于 A。对于此类小问题，精简分解只是略小一些。

```
[U,S,V] = svd(A,0)
```

```
U =
```

```
0.6105 -0.7174  
0.6646  0.2336  
0.4308  0.6563
```

```
S =
```

```
14.9359      0  
0   5.1883
```

```
V =
```

```
0.6925 -0.7214  
0.7214  0.6925
```

同样， $U \cdot S \cdot V'$ 在舍入误差界限内等于 A。

如果矩阵 A 很大并且是稀疏矩阵，则使用 svd 来计算所有奇异值和向量在某些情况下可能会不太切合实际。例如，如果您只需了解几个最大的奇异值，则计算一个 5000×5000 稀疏矩阵的所有奇异值会带来大量额外工作。在只需要一部分奇异值和向量的情况下，svds 函数优先于 svd。

对于一个密度约为 30% 的 1000×1000 随机稀疏矩阵，

```
n = 1000;  
A = sprand(n,n,0.3);
```

最大的六个奇异值为

```
S = svds(A)
```

```
S =
```

```
130.2184  
16.4358  
16.4119  
16.3688  
16.3242  
16.2838
```

此外，最小的六个奇异值为

```
S = svds(A,6,'smallest')
```

```
S =
```

```
0.0740  
0.0574  
0.0388  
0.0282  
0.0131  
0.0066
```

对于可作为满矩阵 `full(A)` 载入内存的较小矩阵，使用 `svd(full(A))` 的速度可能仍旧快于使用 `svds`。但对于确实很大的稀疏矩阵，就有必要使用 `svds`。

MATLAB 中的 LAPACK

LAPACK (线性代数包) 是一个例程库，它为数值线性代数和矩阵计算提供快速、稳健的算法。MATLAB 中的线性代数函数和矩阵运算均基于 LAPACK 构建，并且继续受益于其例程的性能和精度。

简史

MATLAB 诞生于 20 世纪 70 年代后期，是一款基于 LINPACK 和 EISPACK 构建的交互式计算器，而 LINPACK 和 EISPACK 在当时是进行矩阵计算的最先进的 Fortran 子例程库。多年来，MATLAB 使用了 LINPACK 和 EISPACK 的十几个 Fortran 子例程的 C 语言版本。

2000 年，MATLAB 改用 LAPACK，这是 LINPACK 和 EISPACK 的现代替代品。它是一个用于数值线性代数的大型、多作者 Fortran 库。LAPACK 最初是为在超级计算机上使用而设计的，因为它能够一次计算矩阵的多个列。LAPACK 例程的速度与基本线性代数子例程 (BLAS) 的速度密切相关。BLAS 版本通常特定于硬件并经过高度优化。

另请参阅

详细信息

- “调用 LAPACK 和 BLAS 函数”

外部网站

- MATLAB 引入 LAPACK

矩阵指数

此示例说明 19 种矩阵指数计算方法中的 3 种。

有关矩阵指数计算的背景信息，请参阅：

Moler, C. and C. Van Loan."Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later."SIAM Review.Vol. 45, Number 1, 2003, pp. 3-49.

首先创建矩阵 A。

```
A = [0 1 2; 0.5 0 1; 2 1 0]
```

A = 3×3

0	1.0000	2.0000
0.5000	0	1.0000
2.0000	1.0000	0

```
Asave = A;
```

方法 1：加权平方

expmdemo1 是以下著作中算法 11.3.1 的实现：

Golub, Gene H. and Charles Van Loan.Matrix Computations, 3rd edition.Baltimore, MD:Johns Hopkins University Press, 1996.

```
% Scale A by power of 2 so that its norm is < 1/2 .
[f,e] = log2(norm(A,'inf));
s = max(0,e+1);
A = A/2^s;
```

```
% Padé approximation for exp(A)
X = A;
c = 1/2;
E = eye(size(A)) + c*A;
D = eye(size(A)) - c*A;
q = 6;
p = 1;
for k = 2:q
    c = c * (q-k+1) / (k*(2*q-k+1));
    X = A*X;
    cX = c*X;
    E = E + cX;
    if p
        D = D + cX;
    else
        D = D - cX;
    end
    p = ~p;
end
E = D\X;
```

```
% Undo scaling by repeated squaring
for k = 1:s
```

```

E = E*E;
end

E1 = E

E1 = 3×3

5.3091  4.0012  5.5778
2.8088  2.8845  3.1930
5.1737  4.0012  5.7132

```

方法 2：泰勒级数

expmdemo2 使用矩阵指数的经典定义，表示为幂级数

$$e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k.$$

A^0 是与 A 具有相同维度的单位矩阵。作为一种实用的数值方法，如果 **norm(A)** 太大，此方法将很慢且不准确。

```

A = Asave;

% Taylor series for exp(A)
E = zeros(size(A));
F = eye(size(A));
k = 1;

while norm(E+F-E,1) > 0
    E = E + F;
    F = A*F/k;
    k = k+1;
end

E2 = E

E2 = 3×3

5.3091  4.0012  5.5778
2.8088  2.8845  3.1930
5.1737  4.0012  5.7132

```

方法 3：特征值和特征向量

expmdemo3 假定矩阵包含一组完整的特征向量 V ，使得 $A = VDV^{-1}$ 。矩阵指数可以通过对特征值的对角矩阵求幂来计算：

$$e^A = Ve^D V^{-1}.$$

作为一种实际的数值方法，准确性由特征向量矩阵的条件确定。

```

A = Asave;

[V,D] = eig(A);
E = V * diag(exp(diag(D))) / V;

```

E3 = E

E3 = 3×3

5.3091	4.0012	5.5778
2.8088	2.8845	3.1930
5.1737	4.0012	5.7132

比较结果

对于此示例中的矩阵，所有三种方法都同样有效。

E = expm(Asave);
err1 = E - E1

err1 = 3×3
 $10^{-14} \times$

0.3553	0.1776	0.0888
0.0888	0.1332	-0.0444
0	0	-0.2665

err2 = E - E2

err2 = 3×3
 $10^{-14} \times$

0	0	-0.1776
-0.0444	0	-0.0888
0.1776	0	0.0888

err3 = E - E3

err3 = 3×3
 $10^{-14} \times$

-0.7105	-0.5329	-0.7105
-0.6661	-0.5773	-0.8882
-0.7105	-0.7105	-0.9770

泰勒级数失败

对于某些矩阵，泰勒级数中的项在变为零之前变得非常大。因此，**expmdemo2** 失败。

A = [-147 72; -192 93];
E1 = expmdemo1(A)

E1 = 2×2

-0.0996	0.0747
-0.1991	0.1494

E2 = expmdemo2(A)

E2 = 2×2
 $10^6 \times$

-1.1985 -0.5908
-2.7438 -2.0442

E3 = expmdemo3(A)

E3 = 2×2

-0.0996 0.0747
-0.1991 0.1494

特征值和特征向量失败

以下是不包含一组完整的特征向量的矩阵。因此，**expmdemo3** 失败。

A = [-1 1; 0 -1];

E1 = expmdemo1(A)

E1 = 2×2

0.3679 0.3679
0 0.3679

E2 = expmdemo2(A)

E2 = 2×2

0.3679 0.3679
0 0.3679

E3 = expmdemo3(A)

E3 = 2×2

0.3679 0
0 0.3679

另请参阅

expm

指数函数的图形比较

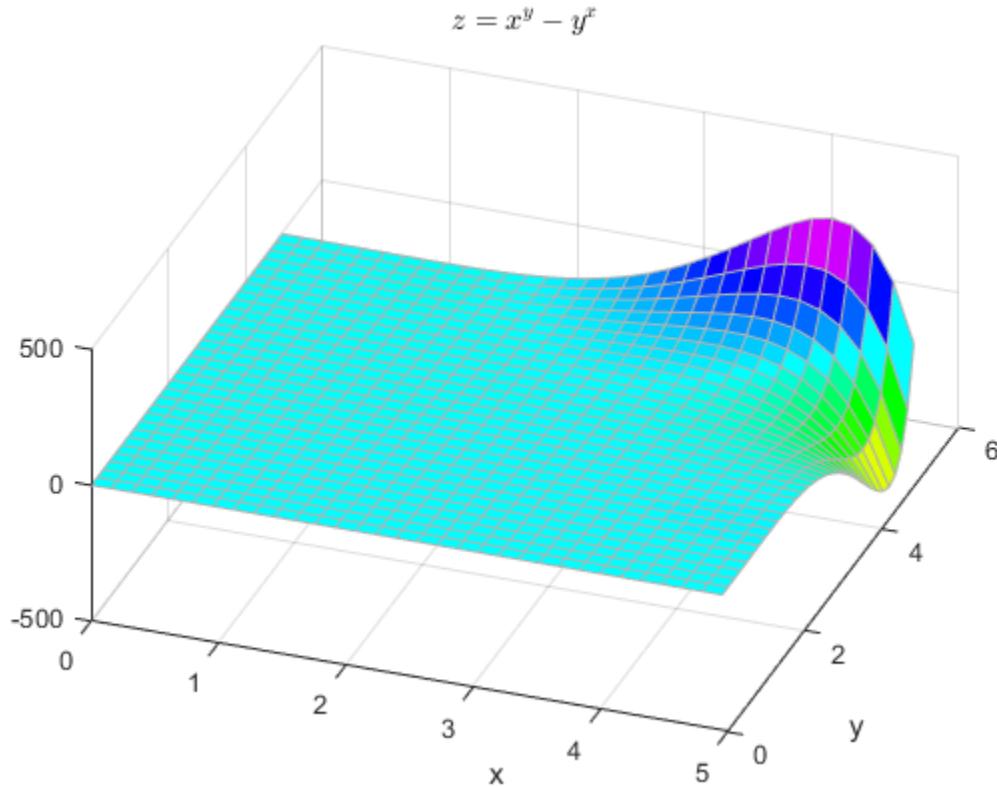
此示例介绍了一种有趣的图形方法，用以确定 e^π 是否大于 π^e 。

问题： e^π 和 π^e 哪一个更大？最简单的确定方法是直接通过 MATLAB® 命令提示符键入这两个值。但是，另一种分析方法是提出一个更普遍的问题：函数 $z(x, y) = x^y - y^x$ 是什么形状？

下面是 z 的图。

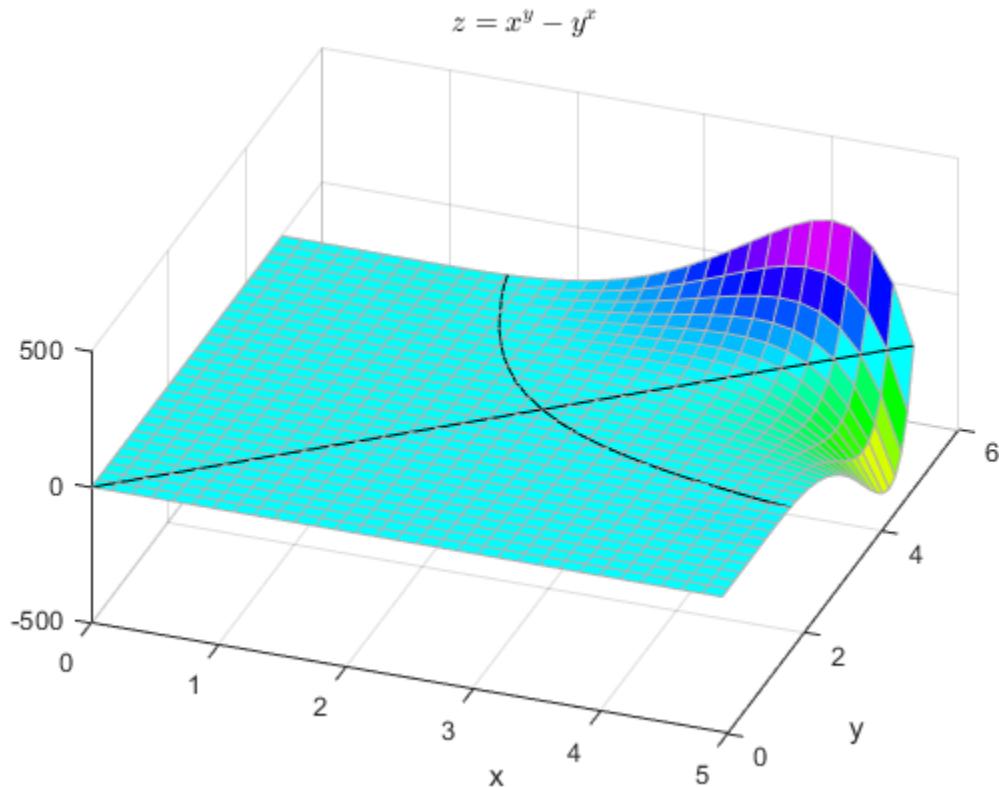
```
% Define the mesh
x = 0:0.16:5;
y = 0:0.16:5;
[xx,yy] = meshgrid(x,y);

% The plot
zz = xx.^yy-yy.^xx;
h = surf(x,y,zz);
h.EdgeColor = [0.7 0.7 0.7];
view(20,50);
colormap(hsv);
title('$z = x^y - y^x$','Interpreter','latex')
xlabel('x')
ylabel('y')
hold on
```



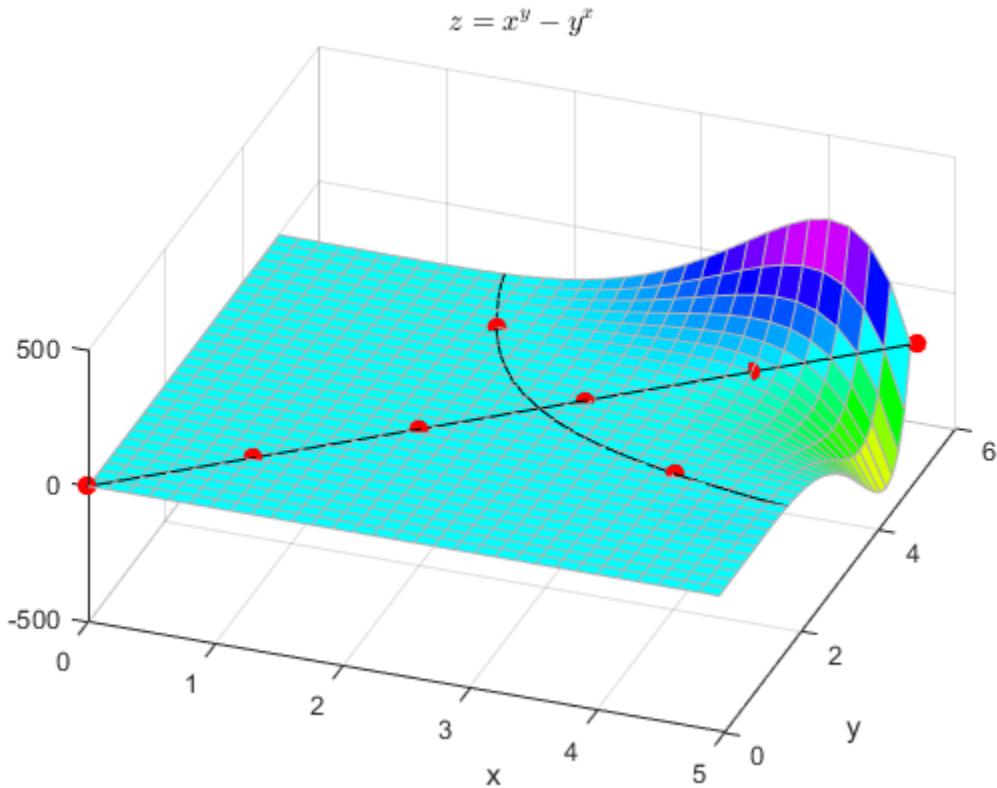
$x^y - y^x = 0$ 方程解的形状较为特别，单纯通过观察并不足以解决我们一开始的问题。下图是使得 $z = 0$ 的 xy 值。

```
c = contourc(x,y,zz,[0 0]);
list1Len = c(2,1);
xContour = [c(1,2:1+list1Len) NaN c(1,3+list1Len:size(c,2))];
yContour = [c(2,2:1+list1Len) NaN c(2,3+list1Len:size(c,2))];
% Note that the NAN above prevents the end of the first contour line from being
% connected to the beginning of the second line
line(xContour,yContour,'Color','k');
```



黑色曲线上部分点处的 x 和 y 同时为整数。下图是方程 $x^y - y^x = 0$ 的整数解。请注意， $2^4 = 4^2$ 是 $x \neq y$ 时的唯一整数解。

```
plot([0:5 2 4],[0:5 4 2],'r','MarkerSize',25);
```

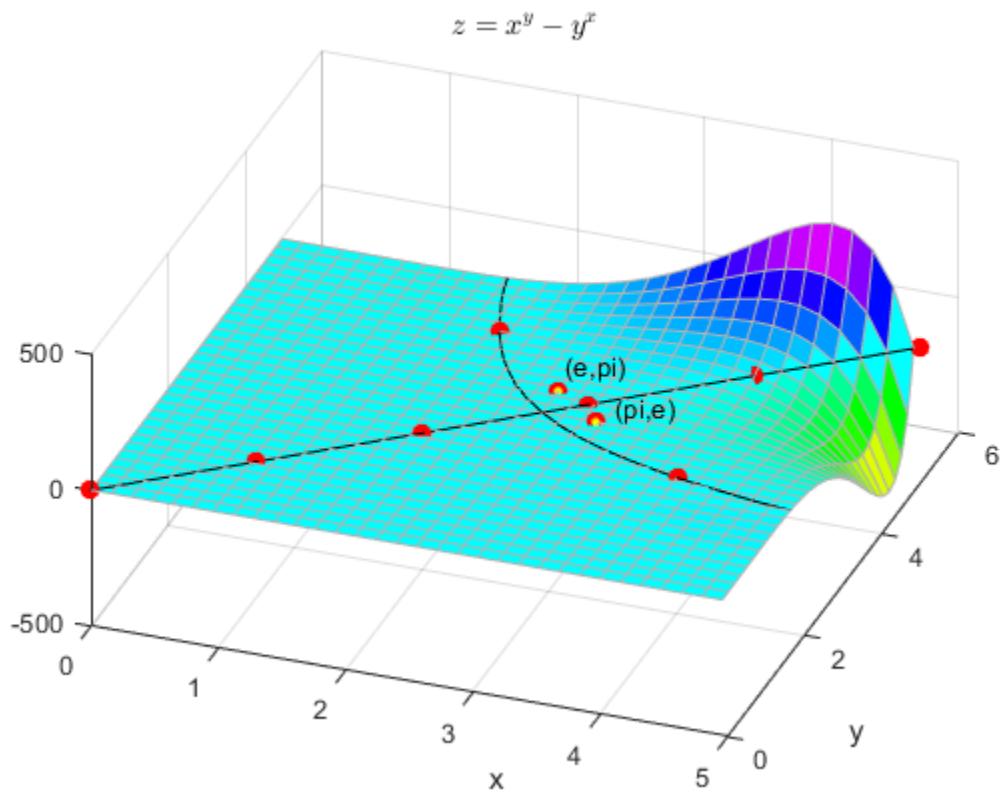


最后，在曲面上绘制点 (π, e) 和 (e, π) 。结果显示， e^π 确实大于 π^e （尽管相差不大）。

```

e = exp(1);
plot([e pi],[pi e],'r','MarkerSize',25);
plot([e pi],[pi e],'y','MarkerSize',10);
text(e,3.3,'(e,pi)','Color','k',...
    'HorizontalAlignment','left','VerticalAlignment','bottom');
text(3.3,e,'(pi,e)','Color','k','HorizontalAlignment','left',...
    'VerticalAlignment','bottom');
hold off;

```



验证结果。

```
e = exp(1);  
e^pi
```

```
ans = 23.1407
```

```
pi^e
```

```
ans = 22.4592
```

另请参阅

[exp](#) | [pi](#)

基本矩阵运算

以下示例演示了以 MATLAB® 语言处理矩阵的基本方法和函数。

首先，创建一个名为 `a` 且包含 9 个元素的简单向量。

```
a = [1 2 3 4 6 4 3 4 5]
```

```
a = 1×9
```

```
1 2 3 4 6 4 3 4 5
```

现在，对向量 `a` 中的每个元素加 2，并将结果存储在一个新向量中。

请注意 MATLAB 不需要对向量或矩阵运算进行特殊的处理。

```
b = a + 2
```

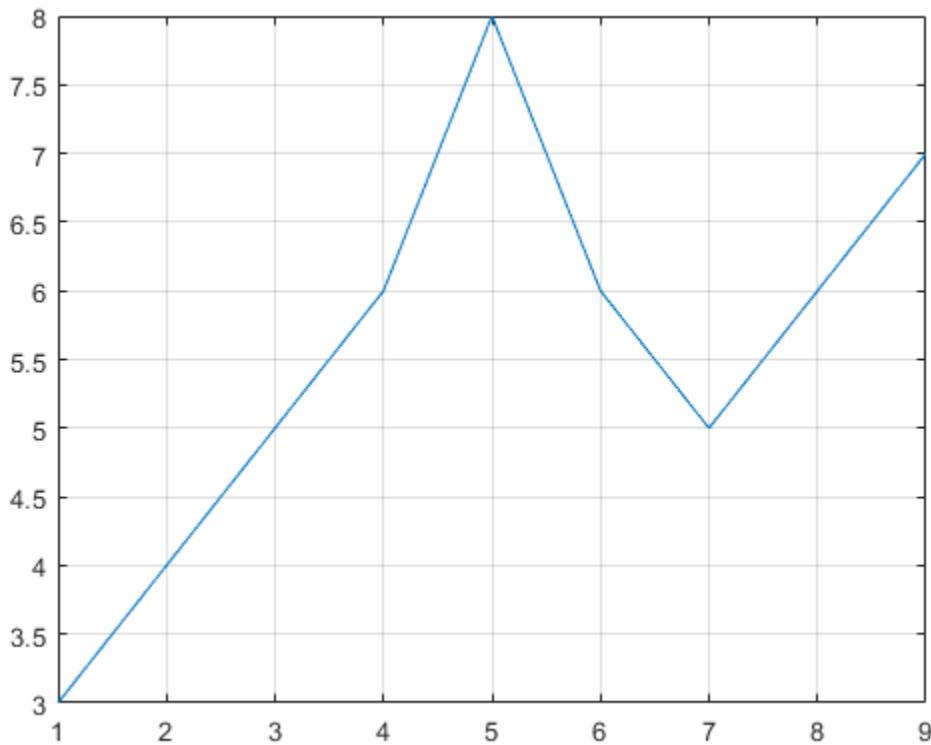
```
b = 1×9
```

```
3 4 5 6 8 6 5 6 7
```

在 MATLAB 中创建图形就像执行一条命令一样简单。接下来用网格线来绘制向量和结果。

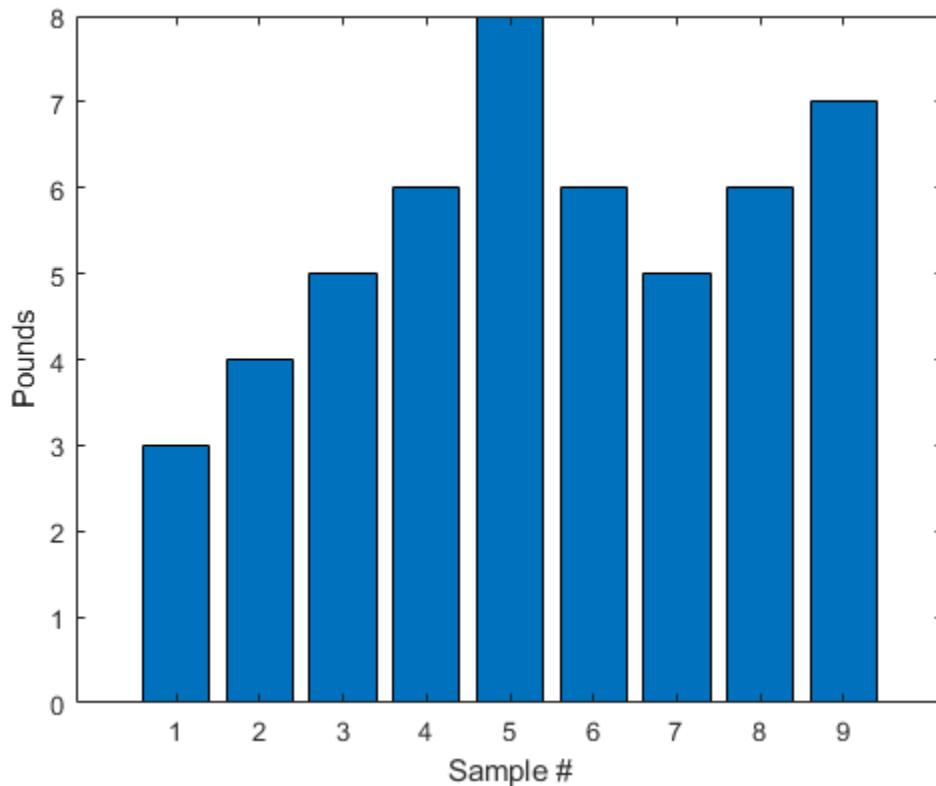
```
plot(b)
```

```
grid on
```



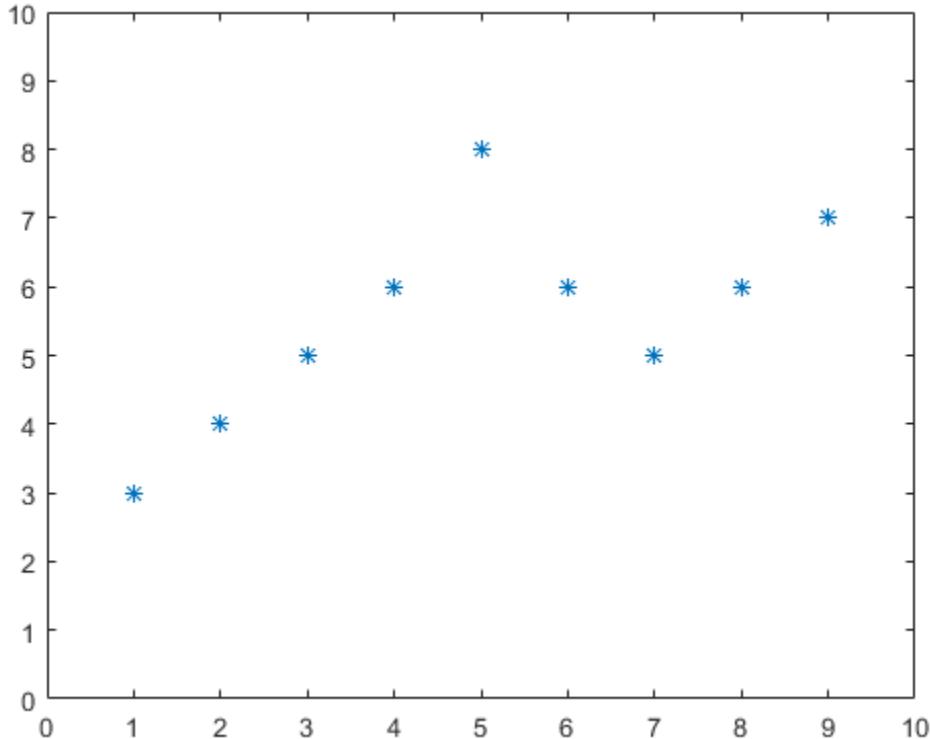
MATLAB 也可以创建包含轴标签的其他图表类型。

```
bar(b)
xlabel('Sample #')
ylabel('Pounds')
```



MATLAB 也可以在绘图中使用符号。下面是用星号来标记各个点的一个示例。MATLAB 提供了多种符号和线型。

```
plot(b, '*')
axis([0 10 0 10])
```



MATLAB 擅长的一个方面是矩阵计算。

创建矩阵就像创建向量一样简单，可使用分号 (;) 分隔矩阵的各行。

A = [1 2 0; 2 5 -1; 4 10 -1]

A = 3×3

```
1   2   0
2   5  -1
4  10  -1
```

可以很容易地计算矩阵 A 的转置。

B = A'

B = 3×3

```
1   2   4
2   5  10
0  -1  -1
```

接下来，将这两个矩阵相乘。

同样请注意，MATLAB 不要求像处理数据集合一样处理矩阵。MATLAB 知道您正在处理矩阵并相应调整您的计算。

C = A * B

C = 3×3

```
5 12 24  
12 30 59  
24 59 117
```

无需执行矩阵相乘，使用 .* 运算符即可将两个矩阵或向量的对应元素相乘。

C = A .* B

C = 3×3

```
1 4 0  
4 25 -10  
0 -10 1
```

使用矩阵 A 对方程 $A^*x = b$ 求解，方法是使用 \ (反斜杠) 运算符。

b = [1;3;5]

b = 3×1

```
1  
3  
5
```

x = A\b

x = 3×1

```
1  
0  
-1
```

现在可以显示 A^*x 等于 b。

r = A*x - b

r = 3×1

```
0  
0  
0
```

MATLAB 拥有几乎所有用于常见矩阵计算的函数。

有用于获取特征值的函数...

eig(A)

ans = 3×1

3.7321

```
0.2679
1.0000
```

...以及用于获取奇异值的函数。

svd(A)

```
ans = 3×1
```

```
12.3171
0.5149
0.1577
```

“poly” 函数生成特征多项式系数的向量。

矩阵 A 的特征多项式为

$$\det(\lambda I - A)$$

p = round(poly(A))

```
p = 1×4
```

```
1   -5   5   -1
```

使用 roots 函数很容易确定多项式的根。

这些值实际上是原始矩阵的特征值。

roots(p)

```
ans = 3×1
```

```
3.7321
1.0000
0.2679
```

除了矩阵计算之外，MATLAB 还有许多其他应用。

求两个向量的卷积...

q = conv(p,p)

```
q = 1×7
```

```
1   -10   35   -52   35   -10   1
```

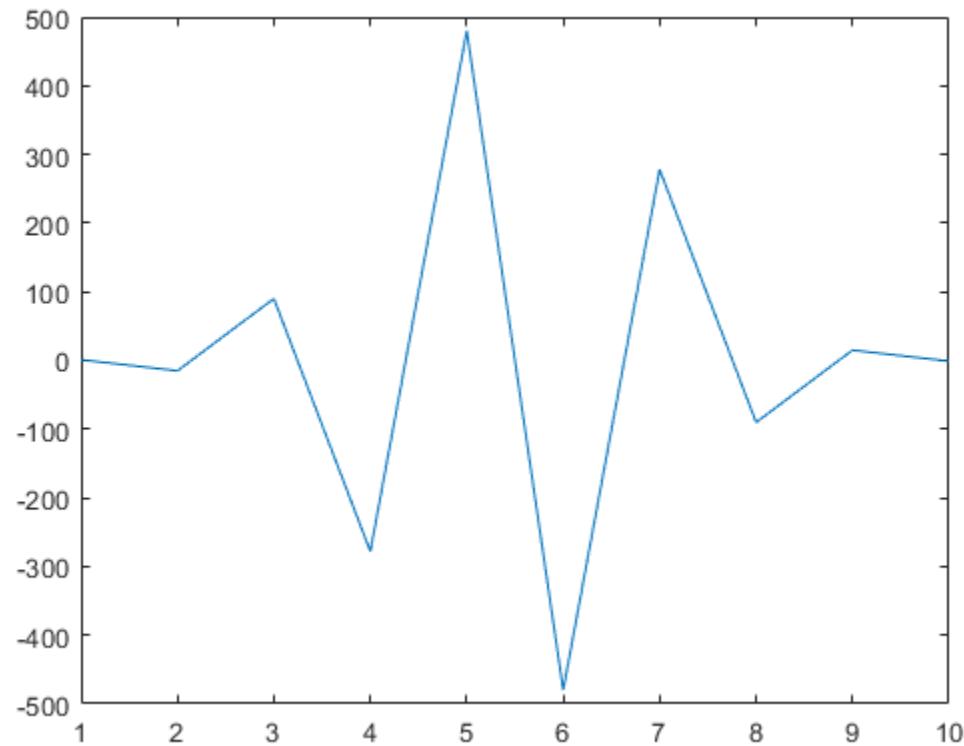
...或再次求卷积并绘制结果图。

r = conv(p,q)

```
r = 1×10
```

```
1   -15   90   -278   480   -480   278   -90   15   -1
```

```
plot(r);
```



使用 **who** 或 **whos** 命令可随时获取内存中存储的变量列表。

whos

Name	Size	Bytes	Class	Attributes
A	3x3	72	double	
B	3x3	72	double	
C	3x3	72	double	
a	1x9	72	double	
ans	3x1	24	double	
b	3x1	24	double	
p	1x4	32	double	
q	1x7	56	double	
r	1x10	80	double	
x	3x1	24	double	

可以通过键入特定变量的名称来获取该变量的值。

A

A = 3×3

```
1   2   0
2   5  -1
4  10  -1
```

每行可以有一条以上的语句，使用逗号或分号分隔各个语句。

如果未指定变量来存储操作的结果，则结果将存储在名为 `ans` 的临时变量中。

```
sqrt(-1)
```

```
ans = 0.0000 + 1.0000i
```

正如所见，MATLAB 在计算中很容易处理复数。

另请参阅

详细信息

- “数组与矩阵运算”

判断矩阵是否为对称正定矩阵

本主题介绍如何使用 `chol` 和 `eig` 函数来确定矩阵是否为对称正定矩阵（特征值全为正的对称矩阵）。

方法 1：尝试 Cholesky 分解

检查矩阵是否为对称正定矩阵的最有效方法是简单地尝试对矩阵使用 `chol`。如果分解失败，则矩阵不是对称正定矩阵。此方法不要求矩阵为对称矩阵也能成功进行测试（如果矩阵不对称，则分解将会失败）。

```
A = [1 -1 0; -1 5 0; 0 0 7]
```

```
A = 3×3
```

```
1   -1   0
-1   5   0
 0   0   7
```

```
try chol(A)
    disp('Matrix is symmetric positive definite.')
catch ME
    disp('Matrix is not symmetric positive definite')
end
```

```
ans = 3×3
```

```
1.0000  -1.0000      0
 0    2.0000      0
 0      0    2.6458
```

```
Matrix is symmetric positive definite.
```

这种方法的缺点是它不能扩展到也检查矩阵是否为对称半正定矩阵（特征值可以为正或零）。

方法 2：检查特征值

虽然使用 `eig` 来计算所有特征值并检查其值效率较低，但是这种方法更灵活，因为您也可以用它来检查矩阵是否为对称半正定矩阵。不过，对于小矩阵来说，检查矩阵是否为对称正定矩阵的这两种方法的计算时间之间的差异可以忽略。

此方法要求您在执行测试之前，先使用 `issymmetric` 来检查矩阵是否对称（如果矩阵不对称，则不需要计算特征值）。

```
tf = issymmetric(A)
```

```
tf = logical
 1
```

```
d = eig(A)
```

```
d = 3×1
```

```
0.7639
5.2361
7.0000
```

```
isposdef = all(d > 0)
```

```
isposdef = logical  
1
```

使用命令 `all(d >= 0)`, 您可以将此方法扩展到检查矩阵是否为对称半正定矩阵。

数值注意事项

此处概述的方法可能对同一矩阵给出不同结果。由于这两种计算都涉及舍入误差，因此每种算法检查有定性时基于的矩阵与 A 有微小的差异。在实践中，使用容差是一种更稳健的比较方法，因为特征值可能在计算机精度范围内表示为数值零，但实际上稍大于零的正数，或稍小于零的负数。

例如，如果矩阵具有 `eps` 数量级的特征值，则使用比较 `isposdef = all(d > 0)` 会返回 `true`，即使特征值在数值上为零，矩阵也更适宜分类为对称半正定矩阵。

要使用容差执行比较，可以使用以下经过修正的命令

```
tf = issymmetric(A)  
d = eig(A)  
isposdef = all(d > tol)  
issemdif = all(d > -tol)
```

容差定义以零为中心的半径，在该半径内的任何特征值都视为零。在大多数情况下，容差为 `length(d)*eps(max(d))` 都是合适的，它考虑了最大特征值的模。

另请参阅

`chol` | `eig`

详细信息

- “分解” (第 2-19 页)

随机数

- “随机数为什么可在启动后重复出现？”（第 3-2 页）
- “创建随机数数组”（第 3-3 页）
- “特定范围内的随机数”（第 3-5 页）
- “随机整数”（第 3-6 页）
- “具有特定均值和方差的正态分布随机数”（第 3-7 页）
- “球体内的随机数”（第 3-8 页）
- “生成可重复的随机数”（第 3-10 页）
- “生成不同的随机数”（第 3-13 页）
- “管理全局流”（第 3-14 页）
- “创建和控制随机数流”（第 3-19 页）
- “多个流”（第 3-24 页）
- “更换不推荐的 rand 和 randn 语法”（第 3-26 页）
- “控制随机数的生成”（第 3-29 页）

随机数为什么可在启动后重复出现？

所有随机数函数（`rand`、`randn`、`randi` 和 `randperm`）均可从共享随机数生成器中抽取值。每次启动 MATLAB 时，生成器均复位到相同的状态。因此，当启动后立即执行计算的任何时候，类似 `rand(2,2)` 的命令均返回相同的结果。此外，无论何时重新启动，任何调用随机数函数的脚本或函数均返回相同的结果。

如果希望在重启 MATLAB 时避免重复相同的随机数数组，必须先执行以下命令：

```
rng('shuffle');
```

然后再调用 `rand`、`randn`、`randi` 或 `randperm`。此命令可确保不会重复 MATLAB 以前会话的结果。

如果想在不重启 MATLAB 会话的情况下重复该会话开始时获得的结果，可随时使用以下命令将生成器重置为启动状态：

```
rng('default');
```

在执行 `rng('default')` 时，后续的随机数命令将返回与新的 MATLAB 会话输出相匹配的结果。例如，

```
rng('default');
A = rand(2,2)
```

```
A =
```

```
0.8147 0.1270
0.9058 0.9134
```

无论何时重启 MATLAB 时，`A` 中的值始终与 `rand(2,2)` 的输出相匹配。

另请参阅

`rng`

创建随机数数组

本节内容

- “随机数函数”（第 3-3 页）
- “随机数生成器”（第 3-3 页）

MATLAB 使用算法来生成伪随机数和伪独立数。这些数在数学意义上并非严格随机和独立的，但它们能够通过各种随机和独立统计测试，并且其计算可以重复，方便用于测试或诊断目的。

rand、**randi**、**randn** 和 **randperm** 函数是创建随机数数组的主要函数。**rng** 函数允许您控制生成随机数的种子和算法。

随机数函数

有四种基本随机数函数：**rand**、**randi**、**randn** 和 **randperm**。**rand** 函数返回在 0 和 1 之间均匀分布的实数。例如，

```
r1 = rand(1000,1);
```

r1 是一个含有均匀分布浮点实数的 1000×1 的列向量。**r1** 中的所有值均处于开区间 (0, 1) 内。这些值的直方图大致上是扁平形状，这表明采样数相当均匀。

randi 函数返回离散均匀分布中的 **double** 整数值。例如，

```
r2 = randi(10,1000,1);
```

r2 是一个包含范围在 1,2,...,10 的离散均匀分布整数值的 1000×1 列向量。这些值的直方图大致上是扁平形状，这表明从 1 到 10 之间采样数相当均匀。

randn 函数返回标准正态分布中的浮点实数数组。例如，

```
r3 = randn(1000,1);
```

r3 是一个含有标准正态分布数的 1000×1 的列向量。**r3** 的直方图看似大致上均值为 0 且标准差为 1 的正态分布。

可以使用 **randperm** 函数创建没有重复值的随机整数值数组。例如，

```
r4 = randperm(15,5);
```

r4 是一个包含在闭区间 [1, 15] 内随机选择的整数值的 1×5 数组。与可返回包含重复值的数组的 **randi** 不同，**randperm** 返回的数组没有重复值。

接连调用上述任一函数均可返回不同的结果。这种特性适合用于创建几个不同的随机值数组。

随机数生成器

MATLAB 提供几个生成器算法选项，下表对其进行了总结。

关键字	生成器	多流和子流支持	全精度的近期周期
mt19937ar	梅森旋转 (MATLAB 启动时默认流使用的算法)	否	$2^{19937}-1$

关键字	生成器	多流和子流支持	全精度的近期周期
dsfmt19937	面向 SIMD 的快速梅森旋转算法	否	$2^{19937}-1$
mcg16807	乘法同余生成器	否	$2^{31}-2$
mlfg6331_64	乘法滞后 Fibonacci 生成器	是	2^{124} (2^{51} 个流, 长度为 2^{72})
mrg32k3a	组合多递归生成器	是	2^{191} (2^{63} 个流, 长度为 2^{127})
philox4x32_10	执行 10 轮的 Philox 4×32 生成器	是	2^{193} (2^{64} 个流, 长度为 2^{129})
threefry4x64_20	执行 20 轮的 Threefry 4×64 生成器	是	2^{514} (2^{256} 个流, 长度为 2^{258})
shr3cong	移位寄存器生成器与线性同余生成器求和	否	2^{64}
swb2712	修正的借位减法生成器	否	2^{1492}

使用 `rng` 函数可设置 `rand`、`randi`、`randn` 和 `randperm` 函数使用的种子和生成器。例如，`rng('shuffle','philox')` 根据当前时间为 Philox 4×32 生成器设置种子，每次调用它时都会产生一个不同数字序列。

有关详细信息，请参阅“控制随机数的生成”（第 3-29 页）。

另请参阅

`rand` | `randi` | `randn` | `randperm` | `rng`

相关示例

- “控制随机数的生成”（第 3-29 页）
- “生成可重复的随机数”（第 3-10 页）
- “生成不同的随机数”（第 3-13 页）
- “特定范围内的随机数”（第 3-5 页）
- “随机整数”（第 3-6 页）
- “具有特定均值和方差的正态分布随机数”（第 3-7 页）

特定范围内的随机数

本示例显示如何基于开区间 (50, 100) 内的均匀分布创建随机浮点数组。

在默认情况下，**rand** 返回均匀分布的归一化值 (0 和 1 之间)。要改变分布的范围 (a、b)，应将各值乘以新范围的宽度 (b - a)，然后用 a 替换各值。

首先，初始化随机数生成器，以使本示例中的结果具备可重复性。

```
rng(0,'twister');
```

创建一个 1000 个随机值的向量。使用 **rand** 函数从开区间 (50,100) 抽取均匀分布的值。

```
a = 50;  
b = 100;  
r = (b-a).*rand(1000,1) + a;
```

验证 r 中的数值是否在指定范围内。

```
r_range = [min(r) max(r)]
```

```
r_range =
```

```
50.0261 99.9746
```

结果是处于开区间 (50、100) 内。

注意 a 和 b 的某些组合在理论上可使得到的结果包括 a 或 b。在实际中，这种情况极难出现。

另请参阅

rng

相关示例

- “具有特定均值和方差的正态分布随机数” (第 3-7 页)
- “球体内的随机数” (第 3-8 页)
- “创建随机数数组” (第 3-3 页)

随机整数

此示例说明了如何根据一组数字 -10、-9、...、9、10 的离散均匀分布创建随机整数值数组。

最简单的 `randi` 语法将返回 1 与指定值 `imax` 之间的双精度整数值。要指定其他范围，请结合使用 `imin` 和 `imax` 参数。

首先，初始化随机数生成器，以使本示例中的结果具备可重复性。

```
rng(0,'twister');
```

基于数字集 -10、-9、...、9、10 的离散均匀分布创建 1×1000 随机整数值数组。使用语法 `randi([imin imax],m,n)`。

```
r = randi([-10 10],1,1000);
```

验证 `r` 中的值是否在指定范围内。

```
r_range = [min(r) max(r)]
```

```
r_range = 1×2
```

```
-10    10
```

另请参阅

`randi` | `rng`

相关示例

- “创建随机数数组”（第 3-3 页）

具有特定均值和方差的正态分布随机数

此示例说明如何基于均值为 500 和方差为 25 的正态分布创建随机浮点数组。

randn 函数返回一个均值为 0 和方差为 1 的正态分布随机数样本。随机变量的一般理论规定，如果 x 是随机变量，其均值是 μ_x 且方差是 σ_x^2 ，则由 $y = ax + b$, (其中 a 和 b 为常量) 定义的随机变量 y 有均值 $\mu_y = a\mu_x + b$ 和方差 $\sigma_y^2 = a^2\sigma_x^2$. 根据此概念，可获得均值为 500 和方差为 25 的正态分布随机数样本。

首先，初始化随机数生成器，以使本示例中的结果具备可重复性。

```
rng(0,'twister');
```

基于均值为 500 且标准差为 5 的正态分布创建包含 1000 个随机值的向量。

```
a = 5;
b = 500;
y = a.*randn(1000,1) + b;
```

计算样本均值、标准差和方差。

```
stats = [mean(y) std(y) var(y)]
```

```
stats = 1×3
```

```
499.8368 4.9948 24.9483
```

均值和方差并非恰好为 500 和 25，因为它们是从分布采样计算得出的。

另请参阅

randn | rng

相关示例

- “特定范围内的随机数” (第 3-5 页)
- “球体内的随机数” (第 3-8 页)
- “创建随机数数组” (第 3-3 页)

球体内的随机数

此示例说明如何按照 Knuth [1] 所述的方法在球体内创建随机点。本例中的球体以原点为中心，半径为 3。

在球体内创建点的方法之一是在球面坐标中指定这些点。然后可以将其转换为笛卡尔坐标进行绘图。

首先，初始化随机数生成器，以使本示例中的结果具备可重复性。

```
rng(0,'twister')
```

计算球体中各个点的仰角。这些值在开区间 $(-\pi/2, \pi/2)$ 内，但不是均匀分布的。

```
rvals = 2*rand(1000,1)-1;  
elevation = asin(rvals);
```

为球体中的各个点创建方位角。这些值在开区间 $(0, 2\pi)$ 内均匀分布。

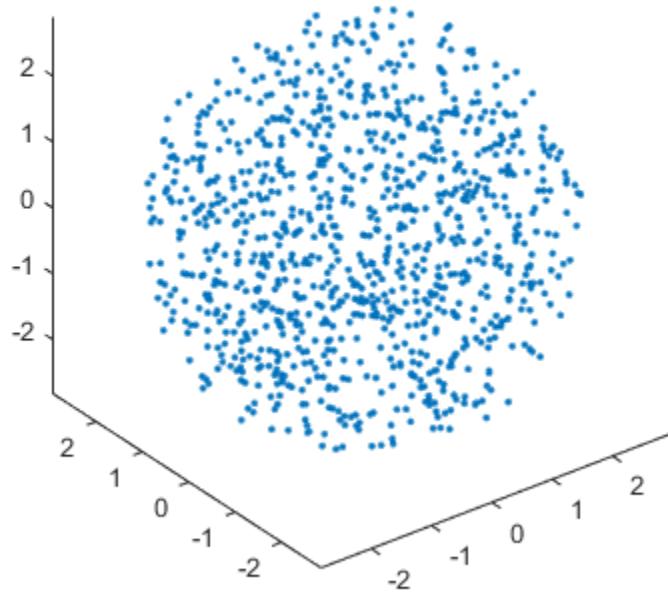
```
azimuth = 2*pi*rand(1000,1);
```

为球体中的各个点创建半径值。这些值在开区间 $(0, 3)$ 内，但不是均匀分布的。

```
radii = 3*(rand(1000,1).^(1/3));
```

转换为笛卡尔坐标并绘制结果。

```
[x,y,z] = sph2cart(azimuth,elevation,radii);  
figure  
plot3(x,y,z,'.')  
axis equal
```



如果需要将随机数置于球体表面上，则指定一个常量半径值作为 `sph2cart` 的最后一个输入参数。在本例中，该值为 3。

```
[x,y,z] = sph2cart(azimuth,elevation,3);
```

参考

[1] Knuth, D. *The Art of Computer Programming*. Vol. 2, 3rd ed. Reading, MA: Addison-Wesley Longman, 1998, pp. 134–136.

另请参阅

[rand](#) | [rng](#) | [sph2cart](#)

相关示例

- “特定范围内的随机数”（第 3-5 页）
- “具有特定均值和方差的正态分布随机数”（第 3-7 页）
- “创建随机数数组”（第 3-3 页）

生成可重复的随机数

指定种子

本示例显示如何通过首先指定种子来重复生成随机数数组。每次使用相同种子初始化生成器时，始终都可以获得相同的结果。

首先，初始化随机数生成器，以使本示例中的结果具备可重复性。

```
rng('default');
```

现在使用种子 1 初始化生成器。

```
rng(1);
```

然后创建随机数数组。

```
A = rand(3,3)
```

```
A =
```

```
0.4170 0.3023 0.1863  
0.7203 0.1468 0.3456  
0.0001 0.0923 0.3968
```

重复同样的命令。

```
A = rand(3,3)
```

```
A =
```

```
0.5388 0.2045 0.6705  
0.4192 0.8781 0.4173  
0.6852 0.0274 0.5587
```

第一次调用 **rand** 改变了生成器的状态，所以第二次调用的结果不同。

现在使用之前的种子重新初始化生成器。然后将再次生成第一个矩阵 A。

```
rng(1);  
A = rand(3,3)
```

```
A =
```

```
0.4170 0.3023 0.1863  
0.7203 0.1468 0.3456  
0.0001 0.0923 0.3968
```

在某些情况下，只设定种子并不能保证相同的结果。这是因为代码执行时，随机数函数所用的生成器可能与您所期望的不同。要确保长期可重复性，应同时指定种子和生成器类型。

例如，以下代码将种子设定为 1 并将生成器设置为梅森旋转。

```
rng(1,'twister');
```

当您希望实现下列结果时，应同时设置种子和生成器：

- 应确保如今编写的代码在以后的 MATLAB 版本中运行时该时可返回相同的结果。
- 应确保在使用当前版本时，您在以前 MATLAB 版本中编写的代码可返回相同的结果。
- 在运行他人的随机数代码之后，再重复自己代码的随机数。

请参见 `rng` 参考页来了解可用的生成器列表。

保存和恢复生成器设置

本示例显示如何通过保存和恢复生成器设置来创建可重复随机数数组。需要保存和恢复生成器设置的最常见原因是重现算法或迭代中某一特定点产生的随机数。例如，可以在调试时使用生成器设置作为辅助工具。

首先，初始化随机数生成器，以使本示例中的结果具备可重复性。

```
rng(1,'twister');
```

在结构体 `s` 中保存生成器的设置。

```
s = rng;
```

创建 1 到 10 之间的随机整数值数组。

```
A = randi(10,3,3)
```

```
A = 3×3
```

5	4	2
8	2	4
1	1	4

重复同样的命令。

```
A = randi(10,3,3)
```

```
A = 3×3
```

6	3	7
5	9	5
7	1	6

第一次调用 `randi` 改变了生成器的状态，所以第二次调用的结果不同。

现在，将生成器恢复为在 `s` 中存储的初始状态，并重新生成第一个数组 `A`。

```
rng(s);
A = randi(10,3,3)
```

```
A = 3×3
```

5	4	2
8	2	4
1	1	4

与重新提供种子（该方法会对生成器进行重新初始化）不同，此方法让您能够随时保存和恢复生成器的设置。

另请参阅

`rng`

相关示例

- “生成不同的随机数”（第 3-13 页）
- “控制随机数的生成”（第 3-29 页）

生成不同的随机数

本示例显示在 MATLAB 重新启动时如何避免重复生成相同的随机数数组。当您要将不同的 MATLAB 会话中执行的相同随机数命令结果合并在一起时，这种技术非常有用。

所有随机数函数（`rand`、`randn`、`randi` 和 `randperm`）均可从共享随机数生成器中抽取值。每次启动 MATLAB 时，生成器均复位到相同的状态。因此，当启动后立即执行计算的任何时候，类似 `rand(2,2)` 的命令均返回相同的结果。此外，无论何时重新启动，任何调用随机数函数的脚本或函数均返回相同的结果。

一种获得不同随机数的方法是每次使用不同的种子初始化生成器。这样做可确保不会重复出现上一次会话的结果。

在调用任何随机数函数之前，在 MATLAB 会话执行一次 `rng('shuffle')` 命令。

```
rng('shuffle')
```

可以在 MATLAB 命令行窗口中执行此命令，也可以将其添加到启动文件中，这是 MATLAB 每次重新启动时都要执行的特殊脚本。

现在执行随机数命令。

```
A = rand(2,2);
```

在每次调用 `rng('shuffle')` 时，它都根据当前的时间使用不同的种子重新设定生成器的种子。

再者，可以明确地指定不同的种子。例如，

```
rng(1);
A = rand(2,2);
rng(2);
B = rand(2,2);
```

数组 A 和 B 不同，因为在每次调用 `rand` 函数之前，都会用不同的种子对生成器进行初始化。

注意 频繁地重新设定生成器的种子既不会改进输出的统计特性，也不会在真正意义上使输出更加随机。在重启 MATLAB 时或在运行涉及随机数的大型计算之前重新设定种子非常有用。然而，在一个会话中过于频繁地重新设定生成器的种子并不是理想的做法，因为随机数的统计特性会受到不利影响。

另请参阅

`rng`

相关示例

- “生成可重复的随机数”（第 3-10 页）
- “控制随机数的生成”（第 3-29 页）
- “在 MATLAB 启动文件中指定启动选项”

管理全局流

rand、**randn** 和 **randi** 从称为全局流的基础随机数流获取随机数。**rng** 函数提供了可控制全局流的简单方法。要进行更全面的控制，可以使用 **RandStream** 类获取全局流句柄并控制随机数生成。

按如下所示获取全局流句柄：

```
globalStream = RandStream.getGlobalStream
globalStream =
mt19937ar random stream (current global stream)
    Seed: 0
    NormalTransform: Ziggurat
```

使用 **get** 方法返回流属性：

```
get(globalStream)
    Type: 'mt19937ar'
    NumStreams: 1
    StreamIndex: 1
    Substream: 1
    Seed: 0
    State: [625x1 uint32]
    NormalTransform: 'Ziggurat'
    Antithetic: 0
    FullPrecision: 1
```

现在，使用 **rand** 函数从全局流生成均匀随机值。

```
rand(1,5);
```

使用 **randn** 和 **randi** 函数从全局流生成正态随机值和整数随机值。

```
A = randi(100,1,5);
A = randn(1,5);
```

State 属性是生成器的内部状态。可以保存 **globalStream** 的 **State**。

```
myState = globalStream.State;
```

使用 **myState**，可以恢复 **globalStream** 的状态并重新生成以前的结果。

```
myState = globalStream.State;
A = rand(1,100);
globalStream.State = myState;
B=rand(1,100);
isequal(A,B)
```

```
ans =
```

```
1
```

rand、**randi** 和 **randn** 访问全局流。由于所有这些函数都访问相同的基础流，因此调用其中一个函数会影响其他函数在后续调用时生成的值。

```
globalStream.State = myState;
A = rand(1,100);
globalStream.State = myState;
```

```
randi(100);
B = rand(1,100);
isequal(A,B)
```

```
ans =
```

```
0
```

全局流是 **RandStream** 类的句柄对象。 **RandStream.getGlobalStream** 返回一个句柄。可以通过全局流的任何句柄来查看或修改该流的属性。

```
stream1=RandStream.getGlobalStream;
stream2=RandStream.getGlobalStream;
stream1.NormalTransform='Polar';
stream2.NormalTransform
ans =
```

Polar

下表显示了可用于 **RandStream** 类的方法。静态方法使用语法 **RandStream.methodName** 进行指示。

方法	说明
RandStream	创建一个随机数流
RandStream.create	创建多个独立的随机数流
get	获取随机流的属性
RandStream.list	列出可用的随机数生成器算法
RandStream.getGlobalStream	获取全局随机数流
RandStream.setGlobalStream	设置全局随机数流
set	设置随机流的属性
reset	将流重置为其初始内部状态
rand	生成均匀分布的伪随机数
randn	生成标准正态分布的伪随机数
randi	生成均匀离散分布的伪随机整数
randperm	随机置换一组值

下表列出了随机流的属性。

属性	说明
Type	(只读) 流使用的生成器算法。 RandStream.list 指定了可能的生成器。
Seed	(只读) 用于创建流的种子值。
NumStreams	(只读) 创建了当前流的组中的流数目。
StreamIndex	(只读) 创建了当前流的流组中的当前流索引。
State	生成器的内部状态。不依赖于此属性的格式。赋给 S.State 的值必须为以前从 S.State 读取的值。

属性	说明
Substream	流当前设置为的子流的索引。默认值为 1。并非所有生成器类型都支持多个子流；乘法滞后 Fibonacci 生成器 (<code>mlfg6331_64</code>) 以及组合多递归生成器 (<code>mrg32k3a</code>) 支持子流。
NormalTransform	<code>randn(s, ...)</code> 生成正态伪随机值时所用的转换算法。可能的值包括 ' <code>Ziggurat</code> '、' <code>Polar</code> ' 或 ' <code>Inversion</code> '。
Antithetic	用于指示 <code>S</code> 是否生成对偶伪随机值的逻辑值。对于均匀值，这些值为从 1 中减去的常见值。默认为 <code>false</code> 。
FullPrecision	逻辑值，用于指示 <code>s</code> 是否使用其全精度来生成值。如果 <code>FullPrecision</code> 为 <code>false</code> ，则某些生成器可以更快地创建伪随机值，但这些值的随机位数更少。默认为 <code>true</code> 。

假定要重复进行仿真。`RandStream` 类提供了多种可复制输出的方法。如前一示例所示，可以保存全局流的状态。

```
myState=GlobalStream.State;
A=rand(1,100);
GlobalStream.State=myState;
B=rand(1,100);
isequal(A,B)

ans =
```

1

还可以使用 `reset` 方法将流重置为其初始设置。

```
reset(GlobalStream)
A=rand(1,100);
reset(GlobalStream)
B=rand(1,100);
isequal(A,B)

ans =
```

1

随机数的数据类型

`rand` 和 `randn` 默认情况下会生成双精度类型的值。

```
GlobalStream=RandStream.getGlobalStream;
myState=GlobalStream.State;
A=rand(1,5);
class(A)
```

`ans =`

`double`

要将类显式指定为双精度类型：

```
GlobalStream.State=myState;
B=rand(1,5,'double');
class(B)
```

```
ans =
```

```
double
isequal(A,B)
```

```
ans =
```

```
1
```

rand 和 **randn** 还会生成单精度类型的值。

```
GlobalStream.State=myState;
A=rand(1,5,'single');
class(A)
ans =
```

```
single
```

这些值与转换前一示例中的双精度值相同。无论返回哪种类的值，这些函数获取的随机流都会以相同方式增加。

```
A,B
```

```
A =
```

```
0.8235 0.6948 0.3171 0.9502 0.0344
```

```
B =
```

```
0.8235 0.6948 0.3171 0.9502 0.0344
```

randi 既支持整数类型，又支持单精度或双精度类型。

```
A=randi([1 10],1,5,'double');
class(A)
```

```
ans =
```

```
double
B=randi([1 10],1,5,'uint8');
class(B)
```

```
ans =
```

```
uint8
```

另请参阅

[rng](#)

相关示例

- “多个流” (第 3-24 页)
- “创建和控制随机数流” (第 3-19 页)
- “控制随机数的生成” (第 3-29 页)
- “创建随机数数组” (第 3-3 页)

创建和控制随机数流

本节内容

- “子流” (第 3-19 页)
- “选择随机数生成器” (第 3-20 页)

通过 **RandStream** 类，可以创建随机数流。在很多情况下，这是很有用的。例如，要生成随机值，而不影响全局流的状态。可能希望在仿真时使用不同的随机性源。或者，可能需要使用与 MATLAB 软件在启动时所用不同的生成器算法。使用 **RandStream** 构造函数，可以创建自己的流、设置可写属性以及生成随机数。可以采用控制全局流的相同方式来控制所创建的流，甚至可将全局流替换为所创建的流。

要创建流，请使用 **RandStream** 构造函数。

```
myStream=RandStream('mlfg6331_64');
rand(myStream,1,5)

ans =
0.6530 0.8147 0.7167 0.8615 0.0764
```

随机流 **myStream** 的行为与全局流的行为不同。函数 **rand**、**randn** 和 **randi** 将继续从全局流中获取，并且不会影响应用于 **myStream** 的 **RandStream** 方法 **rand**、**randn** 和 **randi** 的结果。

可以使用 **RandStream.setGlobalStream** 方法使 **myStream** 成为全局流。

```
RandStream.setGlobalStream(myStream)
RandStream.getGlobalStream

ans =
mlfg6331_64 random stream (current global stream)
    Seed: 0
    NormalTransform: Ziggurat

RandStream.getGlobalStream==myStream

ans =
1
```

子流

也许需要返回到模拟的前一部分。可以通过使随机流跳转到称为子流的固定检查点来控制随机流。**Substream** 属性允许您在多个子流之间来回跳转。要使用 **Substream** 属性，请使用支持子流的生成器创建流。（有关生成器算法及其属性的列表，请参阅“选择随机数生成器”（第 3-20 页）。）

```
stream=RandStream('mlfg6331_64');
RandStream.setGlobalStream(stream)
```

Substream 的初始值为 1。

```
stream.Substream
```

```
ans =
```

1

子流在序列计算中很有用。子流可以通过返回到流中的特定检查点来重新创建所有或部分仿真。例如，可以在循环中使用子流。

```
for i=1:5
    stream.Substream=i;
    rand(1,i)
end

ans =
0.6530

ans =
0.3364 0.8265

ans =
0.9539 0.6446 0.4913

ans =
0.0244 0.5134 0.6305 0.6534

ans =
0.3323 0.9296 0.5767 0.1233 0.6934
```

其中每个子流都可以重新生成其循环迭代。例如，可以返回到第 5 个子流。结果将返回与上面的第 5 个输出相同的值。

```
stream.Substream=5;
rand(1,5)

ans =
0.3323 0.9296 0.5767 0.1233 0.6934
```

选择随机数生成器

MATLAB 提供了多个生成器算法选项。下表概述了可用的生成器算法的关键属性以及用于创建这些算法的关键字。要返回所有可用生成器算法的列表，请使用 `RandStream.list` 方法。

关键字	生成器	多流和子流支持	全精度的近期周期
mt19937ar	梅森旋转 (MATLAB 启动时默认流使用的算法)	否	$2^{19937}-1$
dsfmt19937	面向 SIMD 的快速梅森旋转算法	否	$2^{19937}-1$
mcg16807	乘法同余生成器	否	$2^{31}-2$
mfg6331_64	乘法滞后 Fibonacci 生成器	是	2^{124} (2^{51} 个流，长度为 2^{72})
mrg32k3a	组合多递归生成器	是	2^{191} (2^{63} 个流，长度为 2^{127})

关键字	生成器	多流和子流支持	全精度的近期周期
philox4x32_10	执行 10 轮的 Philox 4×32 生成器	是	2^{193} (2^{64} 个流, 长度为 2^{129})
threefry4x64_20	执行 20 轮的 Threefry 4×64 生成器	是	2^{514} (2^{256} 个流, 长度为 2^{258})
shr3cong	移位寄存器生成器与线性同余生成器求和	否	2^{64}
swb2712	修正的借位减法生成器	否	2^{1492}

生成器 **mcg16807**、**shr3cong** 和 **swb2712** 可与以前的 MATLAB 版本向后兼容。**mt19937ar** 和 **dsmf19937** 主要是为序列应用程序设计的。其余的生成器明确支持并行随机数生成。

根据应用程序情况，某些生成器可能会更快或者返回精度更高的值。所有伪随机数生成器均基于确定性算法，并且均无法通过足够具体的随机性统计学测试。检查蒙特卡罗仿真结果的一种方法是使用两种或更多种不同的生成器算法重新运行该仿真，MATLAB 软件的生成器选项提供了执行该操作的方式。使用不同的生成器时，尽管结果不可能因多个蒙特卡罗采样误差而异，但资料中存于此类验证在特定的生成器算法中出现缺陷的示例（请参阅[13]了解示例）。

生成器算法

mt19937ar

梅森旋转算法（如 [11] 中所述）的周期为 $2^{19937} - 1$ ，并且每个 $U(0,1)$ 值都是使用两个 32 位整数生成的。可能的值为 $(0,1)$ 区间内 2^{-53} 的倍数。此生成器不支持多个流或子流。默认情况下用于 **mt19937ar** 流的 **randn** 算法为 ziggurat 算法 [7]，但基于 **mt19937ar** 生成器。注意：此生成器与 **rand** 函数从 MATLAB 版本 7 开始使用的生成器相同，通过 **rand('twister',s)** 激活。

dsmf19937

面向 SIMD 的双精度快速梅森旋转算法（如 [12] 中所述）是速度更快的梅森旋转算法实现。周期为 $2^{19937} - 1$ ，可能的值为 $(0,1)$ 区间内 2^{-52} 的倍数。生成器本身会生成 $[1,2)$ 内的双精度值，这些值经过变换生成 $U(0,1)$ 值。此生成器不支持多个流或子流。

mcg16807

32 位乘法同余生成器，如 [14] 中所述，其中乘数为 $a = 7^5$ ，模为 $m = 2^{31} - 1$ 。此生成器的周期为 $2^{31} - 2$ ，不支持多个流或子流。每个 $U(0,1)$ 值都是使用单个 32 位整数通过该生成器创建的；可能的值为 $(2^{31} - 1)^{-1}$ 的所有倍数（严格位于区间 $(0,1)$ 内）。**mcg16807** 流默认情况下使用的 **randn** 算法是极坐标算法（如[1]中所述）。注意：此生成器与 **rand** 和 **randn** 函数从 MATLAB 版本 4 开始使用的生成器相同，通过 **rand('seed',s)** 或 **randn('seed',s)** 进行激活。

mlfg6331_64

64 位的乘法滞后 Fibonacci 生成器，如 [10] 中所述，其中滞后值为 $l = 63$ 、 $k = 31$ 。此生成器与 SPRNG 包中实现的 MLFG 类似。其周期约为 2^{124} 。通过参数化，它最多支持 2^{61} 个并行流以及 2^{51} 个长度均为 2^{72} 的子流。每个 $U(0,1)$ 值都是使用一个 64 位整数通过该生成器创建的；可能的值全部为 2^{-64} 的倍数（严格位于区间 $(0,1)$ 内）。默认情况下用于 **mlfg6331_64** 流的 **randn** 算法为 ziggurat 算法 [7]，但基于 **mlfg6331_64** 生成器。

mrg32k3a

32 位组合多递归生成器，如[2]中所述。此生成器与 RngStreams 包中实现的 CMRG 类似。其周期为 2^{191} ，通过序列拆分最多支持 2^{63} 个长度均为 2^{127} 的并行流。它还支持 2^{51} 个长度均为 2^{76} 的子流。每个 $U(0,1)$ 值都是使用两个 32 位整数通过该生成器创建的；可能的值为 2^{-53} 的倍数（严格位于区

间 $(0,1)$ 内)。默认情况下用于 **mrg32k3a** 流的 **randn** 算法为 ziggurat 算法 [7]，但基于 **mrg32k3a** 生成器。

philox4x32_10

执行 10 轮的 4×32 生成器，如 [15] 中所述。此生成器使用 Feistel 网络和整数乘法，专为在高度并行系统（如 GPU）中实现高性能而设计。它的周期为 2^{193} (2^{64} 个流，长度为 2^{129})。

threefry4x64_20

执行 20 轮的 4×64 生成器，如 [15] 中所述。此生成器是 Skein 散列函数中的 Threefish 块加密的非加密改编。它的周期为 2^{514} (2^{256} 个流，长度为 2^{258})。

shr3cong

Marsaglia 的 SHR3 移位寄存器生成器与线性同余生成器求和，其中乘数为 $a = 69069$ ，加数为 $b = 1234567$ ，模为 2^{-32} 。SHR3 是一个定义为 $u = u(\mathbf{I} + \mathbf{L}^{13})(\mathbf{I} + \mathbf{R}^{17})(\mathbf{I} + \mathbf{L}^5)$ 的 3 移位寄存器生成器，其中 \mathbf{I} 为单位运算符， \mathbf{L} 为向左移位运算符， \mathbf{R} 为向右移位运算符。此组合生成器（SHR3 部分在 [7] 中有述）的周期约为 2^{64} 。此生成器不支持多个流或子流。每个 $U(0,1)$ 值都是使用一个 32 位整数通过该生成器创建的；可能的值为 2^{-32} 的所有倍数（严格位于区间 $(0,1)$ 内）。默认情况下用于 **shr3cong** 流的 **randn** 算法为早期形式的 ziggurat 算法 [9]，但基于 **shr3cong** 生成器。此生成器与 **randn** 函数从 MATLAB 版本 5 开始使用的生成器相同，通过 **randn('state',s)** 进行激活。

注意 [6] (1999) 中用到的 SHR3 生成器与 [7] (2000) 中不同。MATLAB 使用 [7] 中提供的最新版生成器。

swb2712

修正的借位减法生成器，如[8]中所述。此生成器与加法滞后 Fibonacci 生成器（其滞后值为 27 和 12）类似，但修改后具有约为 2^{1492} 的更长周期。此生成器本身会使用双精度类型来创建 $U(0,1)$ 值，并且开区间 $(0,1)$ 中的所有值都是可能的。默认情况下用于 **swb2712** 流的 **randn** 算法为 ziggurat 算法 [7]，但基于 **swb2712** 生成器。注意：此生成器与 MATLAB 版本 5 开始的版本中的 **rand** 函数所用生成器相同，通过 **rand('state',s)** 进行激活。

转换算法

Inversion

通过对均匀的随机变量应用标准正态逆累积分布函数来计算正态随机变量。每个正态值恰好使用一个均匀值。

Polar

极坐标抑制算法，如[1]中所述。每个正态值平均约使用 1.27 个均匀值。

Ziggurat

ziggurat 算法，如[7]中所述。每个正态值平均约使用 2.02 个均匀值。

参考

- [1] Devroye, L. *Non-Uniform Random Variate Generation*, Springer-Verlag, 1986.
- [2] L'Ecuyer, P. "Good Parameter Sets for Combined Multiple Recursive Random Number Generators" , *Operations Research*, 47(1): 159–164. 1999.
- [3] L'Ecuyer, P. and S. Côté. "Implementing A Random Number Package with Splitting Facilities" , *ACM Transactions on Mathematical Software*, 17: 98–111. 1991.

- [4] L'Ecuyer, P. and R. Simard. "TestU01: A C Library for Empirical Testing of Random Number Generators," *ACM Transactions on Mathematical Software*, 33(4): Article 22. 2007.
- [5] L'Ecuyer, P., R. Simard, E. J. Chen, and W. D. Kelton. "An Object-Oriented Random-Number Package with Many Long Streams and Substreams." *Operations Research*, 50(6):1073–1075. 2002.
- [6] Marsaglia, G. "Random numbers for C: The END?" Usenet posting to sci.stat.math. 1999. Available online at https://groups.google.com/group/sci.crypt/browse_thread/thread/ca8682a4658a124d/.
- [7] Marsaglia G., and W. W. Tsang. "The ziggurat method for generating random variables." *Journal of Statistical Software*, 5:1–7. 2000. Available online at <https://www.jstatsoft.org/v05/i08>.
- [8] Marsaglia, G., and A. Zaman. "A new class of random number generators." *Annals of Applied Probability* 1(3):462–480. 1991.
- [9] Marsaglia, G., and W. W. Tsang. "A fast, easily implemented method for sampling from decreasing or symmetric unimodal density functions." *SIAM J.Sci.Stat.Comput.* 5(2):349–359. 1984.
- [10] Mascagni, M., and A. Srinivasan. "Parameterizing Parallel Multiplicative Lagged-Fibonacci Generators." *Parallel Computing*, 30: 899–916. 2004.
- [11] Matsumoto, M., and T. Nishimura. "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudorandom Number Generator." *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30. 1998.
- [12] Matsumoto, M., and M. Saito. "A PRNG Specialized in Double Precision Floating Point Numbers Using an Affine Transition." *Monte Carlo and Quasi-Monte Carlo Methods 2008*, 10.1007/978-3-642-04107-5_38. 2009.
- [13] Moler, C.B. *Numerical Computing with MATLAB*. SIAM, 2004. Available online at <https://www.mathworks.com/moler>
- [14] Park, S.K., and K.W. Miller. "Random Number Generators: Good Ones Are Hard to Find." *Communications of the ACM*, 31(10):1192–1201. 1998.
- [15] Salmon, J. K., M. A. Moraes, R. O. Dror, and D. E. Shaw. "Parallel Random Numbers: As Easy as 1, 2, 3." In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC11)*. New York, NY: ACM, 2011.

另请参阅

`rng`

相关示例

- "管理全局流" (第 3-14 页)
- "多个流" (第 3-24 页)
- "控制随机数的生成" (第 3-29 页)
- "创建随机数数组" (第 3-3 页)

多个流

MATLAB 软件提供的生成器算法允许创建多个独立的随机数流。通过 `RandStream.create` 出厂方法，可以创建三个流，这些流具有相同的生成器算法和种子值，但在统计上是独立的。

```
[s1,s2,s3]=RandStream.create('mlfg6331_64','NumStreams',3)
```

```
s1 =
```

```
mlfg6331_64 random stream
  StreamIndex: 1
  NumStreams: 3
  Seed: 0
  NormalTransform: Ziggurat
```

```
s2 =
```

```
mlfg6331_64 random stream
  StreamIndex: 2
  NumStreams: 3
  Seed: 0
  NormalTransform: Ziggurat
```

```
s3 =
```

```
mlfg6331_64 random stream
  StreamIndex: 3
  NumStreams: 3
  Seed: 0
  NormalTransform: Ziggurat
```

作为独立性证据，可以看到这些流在很大程度上是不相关的。

```
r1=rand(s1,100000,1);
r2=rand(s2,100000,1);
r3=rand(s3,100000,1);
corrcoef([r1,r2,r3])
```

```
ans =
```

```
1.0000 -0.0017 -0.0010
-0.0017 1.0000 -0.0050
-0.0010 -0.0050 1.0000
```

通过使用不同的种子，可以创建返回不同的值并且各自独立执行的流。

```
s1=RandStream('mt19937ar','seed',1);
s2=RandStream('mt19937ar','seed',2);
s3=RandStream('mt19937ar','seed',3);
```

种子值必须为介于 0 和 $2^{32} - 1$ 之间的整数。对于不同的种子，流返回的值通常是不相关的。

```
r1=rand(s1,100000,1);
r2=rand(s2,100000,1);
r3=rand(s3,100000,1);
corrcoef([r1,r2,r3])
```

```
ans =
1.0000  0.0030  0.0045
0.0030  1.0000  -0.0015
0.0045  -0.0015  1.0000
```

对于未明确支持独立流的生成器类型，不同的种子提供了可创建多个流的方法。但是，更好的选择是使用专为多个独立的流设计的生成器，因为这样可以更好地了解各流的统计学属性。

根据应用程序的情况，仅创建一组独立流中的部分流可能很有用。StreamIndex 属性返回一组出厂生成的流中的指定流的索引。

```
numLabs=256;
labIndex=4;
s1=RandStream.create('mlfg6331_64',
    'NumStreams',numLabs,'StreamIndices',labIndex)

s1=
mlfg6331_64 random stream
    StreamIndex: 4
    NumStreams: 256
    Seed: 0
    NormalTransform: Ziggurat
```

由于多个流在统计上是独立的，因此可用于验证仿真的精度。例如，可以使用一组独立的流在不同的 MATLAB 会话或不同的处理器中多次重复蒙特卡罗仿真，并确定结果的变化情况。这样使得多个流可用于大规模的并行仿真。

注意 并非所有生成器算法都支持多个流。有关生成器属性的摘要，请参阅“选择随机数生成器”（第 3-20 页）中的生成器算法表。

另请参阅

`rng`

相关示例

- “管理全局流”（第 3-14 页）
- “控制随机数的生成”（第 3-29 页）
- “选择随机数生成器”（第 3-20 页）
- “创建随机数数组”（第 3-3 页）

更换不推荐的 rand 和 randn 语法

本节内容

- “不建议使用的语法说明”（第 3-26 页）
- “替换语法说明”（第 3-26 页）
- “使用整数种子初始化生成器的替换语法”（第 3-27 页）
- “使用状态向量初始化生成器的替换语法”（第 3-27 页）
- “如果无法从不建议使用的语法升级”（第 3-28 页）

不建议使用的语法说明

在早期版本的 MATLAB 中，是通过 'seed'、'state' 或 'twister' 输入来控制 rand 和 randn 函数所用的随机数生成器的。例如：

```
rand('seed',sd)
randn('seed',sd)
rand('state',s)
randn('state',s)
rand('twister',5489)
```

这些语法引用不同类型的生成器，由于下列原因，不建议继续使用这些语法：

- 对于生成器来说，'seed' 和 'state' 的说法具有误导性。
- 所有生成器 ('twister' 除外) 都有缺陷。
- 它们对 rand 和 randn 使用不同的生成器，但这没有必要。

要评估替换您现有代码中不建议使用的语法所带来的影响，请在启动 MATLAB 会话时执行以下命令：

```
warning('on','MATLAB:RandStream:ActivatingLegacyGenerators')
warning('on','MATLAB:RandStream:ReadingInactiveLegacyGeneratorState')
```

替换语法说明

使用 rng 函数可以控制 rand、randn、randi 以及所有其他随机数生成器（如 randperm、sprand 等）使用的共享生成器。要了解在替换不建议使用的语法时如何使用 rng 函数，需要花些时间来了解这些语法的功能。这应该有助于了解哪种新的 rng 语法最适合需要。

rand(Generator,s) 或 **randn(Generator,s)** 的第一个输入指定了生成器的类型，具体如下所述。

Generator = 'seed' 引用的是 MATLAB v4 生成器，而不是种子初始化值。

Generator = 'state' 引用的是 MATLAB v5 生成器，而不是生成器的内部状态。

Generator = 'twister' 引用的是梅森旋转生成器，该生成器现在是 MATLAB 启动时的默认生成器。

除非试图准确地重新生成早期版本的 MATLAB 中生成的随机数，否则不建议再使用 v4 和 v5 生成器。更新代码的最简单方法是使用 rng。rng 函数按如下所示替换 rand 和 randn 所用生成器的名称。

rand/randn 所用生成器的名称	rng 所用生成器的名称
'seed'	'v4'
'state'	'v5uniform' (对于 rand) 或 'v5normal' (对于 randn)
'twister'	'twister' (推荐)

使用整数种子初始化生成器的替换语法

在 `rand(Generator,sd)` 语法中，整数种子 `sd` 的最常见用途是：

- 每次重新生成完全相同的随机数（例如使用 0、1 或 3141879 等种子）
- 尽可能确保 MATLAB 每次运行时总能提供不同的随机数（例如，使用 `sum(100*clock)` 等种子）

下表显示了整数种子为 `sd` 的情况下的语法替换。

- 第一列显示了不建议使用的 `rand` 和 `randn` 语法。
- 第二列显示了如何使用新的 `rng` 函数准确地重现不建议使用的行为。在大多数情况下，这可以通过指定旧生成器类型（例如 v4 或 v5 生成器）来实现，但不建议再使用这种方式。
- 第三列显示了推荐的替代方式，该方式不会为 `rng` 指定可选的生成器类型输入。因此，如果总是忽略 `Generator` 输入，`rand`、`randn` 和 `randi` 将使用默认的梅森旋转生成器，后者是 MATLAB 启动时的默认生成器。在以后的版本中，当新的生成器取代梅森旋转时，此代码将使用新的默认值。

不建议使用的 rand/randn 语法	不推荐：通过指定生成器类型准确重现不推荐的行为	推荐的备用方式：不覆盖生成器类型
<code>rand('twister',5489)</code>	<code>rng(5489,'twister')</code>	<code>rng('default')</code>
<code>rand('seed',sd)</code>	<code>rng(sd,'v4')</code>	<code>rng(sd)</code>
<code>randn('seed',sd)</code>		
<code>rand('state',sd)</code>	<code>rng(sd,'v5uniform')</code>	
<code>randn('state',sd)</code>	<code>rng(sd,'v5normal')</code>	
<code>rand('seed',sum(100*clock))</code>	<code>rng(sum(100*clock),'v4')</code>	<code>rng('shuffle')</code>

使用状态向量初始化生成器的替换语法

在 `rand(Generator,st)` 语法中，状态向量（此处显示为 `st`）的最常见用途是准确地重新生成在算法或迭代的特定点处生成的随机数。例如，可以在调试时使用此向量作为辅助。

`rng` 函数可更改随机数生成器的状态保存和恢复模式，如下表所示。左列中的示例假定正在使用 v5 均匀生成器。右列中的示例使用新的语法，适用于任何生成器。

不建议的 rand/randn 使用语法	使用 rng 的新语法
<pre>% Save v5 generator state. st = rand('state'); % Call rand. x = rand; % Restore v5 generator state. rand('state',st); % Call rand again and hope % for the same results. y = rand</pre>	<pre>% Get generator settings. s = rng; % Call rand. x = rand; % Restore previous generator % settings. rng(s); % Call rand again and % get the same results. y = rand</pre>

有关演示，请观看此说明性视频。

如果无法从不建议使用的语法升级

如果存在无法修改或不允许修改的代码，并且知道该代码使用的是不推荐的随机数生成器控制语法，则务必记住，当使用该代码时，MATLAB 将会切换为旧模式。在旧模式下，**rand** 和 **randn** 由不同的生成器控制，设置各不相同。

在旧模式下调用 **rand** 使用以下生成函数之一：

- 'v4' 生成器（由 **rand('seed', ...)** 控制）
- 'v5uniform' 生成器（由 **rand('state', ...)** 控制）
- 'twister' 生成器（由 **rand('twister', ...)** 控制）

在旧模式下调用 **randn** 使用以下生成函数之一：

- 'v4' 生成器（由 **randn('seed', ...)** 控制）
- 'v5normal' 生成器（由 **randn('state', ...)** 控制）

如果依赖的代码将 MATLAB 置于旧模式，请使用以下命令退出旧模式并返回默认初始生成器：

rng default

此外，要保护将 MATLAB 置于旧模式的代码，请使用：

```
s = rng    % Save current settings of the generator.
...        % Call code using legacy random number generator syntaxes.
rng(s)    % Restore previous settings of the generator.
```

另请参阅

rand | **randn** | **rng**

相关示例

- “创建随机数数组”（第 3-3 页）

控制随机数的生成

此示例说明如何使用 `rng` 函数，该函数针对随机数的生成提供控制。

MATLAB 中的（伪）随机数通过 `rand`、`randi` 和 `randn` 函数生成。许多其他函数调用这三个函数，但这三个函数是基础构建块。这三个函数都依赖于一个共享的随机数生成器，可以使用 `rng` 控制该生成器。

务必注意，MATLAB 中的“随机”数并非是完全不可预测的，而是由确定的算法生成的。该算法设计得极为复杂，这样一来，对不了解该算法的人来说，其输出似乎为独立的随机序列，并且可以通过各种随机性的统计学测试。这里介绍的函数提供了相应方式来利用确定性执行以下操作：

- 重复进行涉及随机数的计算，并获取相同结果，或者
- 保证在重复计算中使用不同的随机数

以及利用明显的随机性来证明不同计算的合并结果。

“从头开始”

如果在新 MATLAB 会话中查看 `rand`、`randi` 或 `randn` 的输出，可注意到，每次重新启动 MATLAB 时，这些函数都返回相同的数字序列。能够将随机数生成器重置到启动状态而实际上并不重新启动 MATLAB，通常很有用。例如，您可能需要重复进行涉及随机数的计算并获取相同结果。

`rng` 提供了一种非常简单的方式来将随机数生成器重置为其默认设置。

```
rng default
rand % returns the same value as at startup
```

```
ans = 0.8147
```

MATLAB 启动时使用的或 `rng default` 为您提供的“默认”随机数设置是什么？如果调用不带任何输入的 `rng`，可以看到，该设置是种子为 0 的梅森旋转生成器算法。

```
rng
ans = struct with fields:
  Type: 'twister'
  Seed: 0
  State: [625x1 uint32]
```

下面详细介绍了如何使用上述输出（包括 `State` 字段）来控制和更改 MATLAB 生成随机数的方式。在上文中，该示例提供了一种方法来查看生成器 `rand`、`randi` 和 `randn` 当前使用的设置。

非重复性

每次调用 `rand`、`randi` 或 `randn` 时，它们都会从其共享的随机数生成器获取一个新值，后续值可被视为在统计上独立的值。但是如上所述，每次重新启动 MATLAB 时，这些函数都将会被重置并返回相同的数字序列。很明显，使用相同“随机”数进行的计算不能认为是与统计无关的。因此，如果需要合并在两个或更多个 MATLAB 会话中完成的计算（好像这些计算是与统计无关的），则无法使用默认生成器设置。

避免在新的 MATLAB 会话中重复生成相同随机数的一种简单方式是，为随机数生成器选择不同的种子。`rng` 通过基于当前时间创建种子，为您提供了一种选择不同种子的简单方式。

```
rng shuffle
rand
```

```
ans = 0.7373
```

每次使用 'shuffle' 时，都将为生成器提供一个不同的种子。可以调用不带任何输入的 `rng` 来查看实际所使用的种子。

`rng`

```
ans = struct with fields:
  Type: 'twister'
  Seed: 1050110405
  State: [625x1 uint32]
```

```
rng shuffle % creates a different seed each time
rng
```

```
ans = struct with fields:
  Type: 'twister'
  Seed: 1050110409
  State: [625x1 uint32]
```

`rand`

```
ans = 0.9301
```

'shuffle' 是为随机数生成器重新提供种子的一种简单方式。您可能认为，使用它在 MATLAB 中实现“真正的”随机性是个好主意，甚至是必要的。但在大多数场合下，完全无需使用 'shuffle'。基于当前时间选择种子并不会改善从 `rand`、`randi` 和 `randn` 获取的值的统计属性，也不会在任何意义上提高这些值的随机性。虽然最好在每次启动 MATLAB 时，或者在运行某种涉及随机数的大型计算之前，都为生成器重新提供种子，但实际上在会话中过于频繁地为生成器重新提供种子并不是一种恰当之举，因为这可能会影响随机数的统计学属性。

'shuffle' 真正提供的是一种避免重复相同值序列的方法。有时它很关键，有时仅是“不错”，但通常完全无非紧要。请记住，如果使用 'shuffle'，您可能需要保存 `rng` 创建的种子，以便在以后重复您的计算。下面说明了如何执行该操作。

进一步控制重复性和非重复性

到此为止，您已了解了如何将随机数生成器重置为其默认设置，以及使用基于当前时间创建的种子为其重新提供种子。`rng` 还提供了一种使用特定种子来为随机数生成器重新提供种子的方法。

您可以多次使用相同的种子来重复进行相同的计算。例如，如果运行下面的代码两次...

```
rng(1) % the seed is any non-negative integer < 2^32
```

```
x = randn(1,5)
```

```
x = 1×5
```

```
-0.6490 1.1812 -0.7585 -1.1096 -0.8456
```

```
rng(1)
```

```
x = randn(1,5)
```

```
x = 1×5
```

```
-0.6490 1.1812 -0.7585 -1.1096 -0.8456
```

...将得到完全一样的结果。通过执行该操作可以在清除 `x` 之后重新创建它，以便在依赖于 `x` 的后续计算中使用这些特定值来重复所要执行的操作。

另一方面，您可能想要选择其他种子来确保不重复相同的计算。例如，如果在某 MATLAB 会话中运行下面的代码...

```
rng(2)
x2 = sum(randn(50,1000),1); % 1000 trials of a random walk
```

并在另一个会话中运行下面的代码...

```
rng(3)
x3 = sum(randn(50,1000),1);
```

...可以合并两个结果，并确信它们不是重复两次的相同结果。

```
x = [x2 x3];
```

与 '`shuffle`' 一样，当为 MATLAB 的随机数生成器重新提供种子时将会出现警告，因为这会影响 `rand`、`randi` 和 `randn` 的所有后续输出。除非需要重复性或唯一性，否则通常建议不要简单通过重新为生成器提供种子来生成随机值。如果的确需要为生成器重新提供种子，通常最好在命令行或代码中不易忽略的某个点上完成。

选择生成器类型

不仅可以按如上所述为随机数生成器重新提供种子，还可以选择要使用的随机数生成器的类型。不同的生成器类型会产生不同的随机数序列，例如，可以因为其统计学属性而选择特定的类型。或者，可能需要从使用不同默认生成器类型的旧版 MATLAB 重新创建结果。

选择生成器类型的另一个常见原因是，您要编写用于生成“随机”输入数据的验证测试，并需要保证测试始终可以得到完全相同的可预测结果。如果在创建输入数据之前调用带种子的 `rng`，这将为随机数生成器重新提供种子。但如果由于某原因更改了生成器类型，则 `rand`、`randi` 和 `randn` 的输出将不会是您期望通过该种子得到的。因此，要 100% 确保可重复性，也可以指定生成器类型。

例如，

```
rng(0,'twister')
```

令 `rand`、`randi` 和 `randn` 采用种子 0 且使用梅森旋转生成器算法。

使用 '`combRecursive`'

```
rng(0,'combRecursive')
```

选择组合多递归生成器算法，该算法支持梅森旋转不支持的某些并行功能。

以下命令

```
rng(0,'v4')
```

选择 MATLAB 4.0 中默认的生成器算法。

当然，此命令会将随机数生成器恢复为其默认设置。

```
rng default
```

但是，由于不同 MATLAB 版本的默认随机数生成器设置可能发生更改，因此使用 '**default**' 并不能长期保证得到可预测的结果。**'default'** 是重置随机数生成器的一种便利方式，但要实现更好的可预测性，请指定生成器类型和种子。

另一方面，在执行交互式工作并且需要可重复性时，调用只带一个种子的 **rng** 不仅更简单，而且通常足够满足要求。

保存并还原随机数生成器的设置

调用不带任何输入的 **rng** 将返回一个标量结构体，其中的字段包含已说明的两条信息：生成器类型和上次为生成器重新提供种子所用的整数。

```
s = rng
s = struct with fields:
  Type: 'twister'
  Seed: 0
  State: [625x1 uint32]
```

第三个字段 **State** 包含生成器的当前状态向量的副本。此状态向量是生成器在内部维护以便在其随机数序列中生成下一个值的信息。每次调用 **rand**、**randi** 或 **randn** 时，它们共享的生成器都会更新其内部状态。因此，**rng** 返回的设置结构体中的状态向量包含从捕获状态的时间点开始重复该序列所需的信息。

虽然可以看到此输出为信息性内容，但 **rng** 也接受设置结构体作为“输入”，以便保存设置（包括状态向量）并在以后还原它们以重复计算。因为设置包含生成器类型，所以可以确切知道将得到的结果，这样“以后”在同一 MATLAB 会话中可能意味着从稍后的某个时刻到以后的几年（和多个 MATLAB 版本）。您可以在随机数序列中从保存生成器设置的任何点重复结果。例如

```
x1 = randn(10,10); % move ahead in the random number sequence
s = rng;           % save the settings at this point
x2 = randn(1,5)

x2 = 1×5
    0.8404   -0.8880    0.1001   -0.5445    0.3035

x3 = randn(5,5); % move ahead in the random number sequence
rng(s);          % return the generator back to the saved state
x2 = randn(1,5) % repeat the same numbers

x2 = 1×5
    0.8404   -0.8880    0.1001   -0.5445    0.3035
```

请注意，虽然重新提供种子仅提供粗略的重新初始化，但使用结构体保存和还原生成器状态，可以重复随机数序列的任意部分。

使用设置结构体的最常见方式是还原生成器状态。但是，由于结构体不仅包含状态，还包含生成器类型和种子，因此临时切换生成器类型也是一种方便的方式。例如，如果需要使用 MATLAB 5.0 中的某个旧生成器创建值，则可以在切换为使用旧生成器的同时保存当前设置...

```
previousSettings = rng(0,'v5uniform')
previousSettings = struct with fields:
  Type: 'twister'
```

```
Seed: 0
State: [625x1 uint32]
```

...然后在以后还原原始设置。

`rng(previousSettings)`

不应修改设置结构体中的任何字段的内容。特别是，不应自行构造状态向量，也不应依赖于生成器状态的格式。

编写更简单、更灵活的代码

通过 `rng`，可以

- 为随机数生成器重新提供种子，或者
- 保存并还原随机数生成器设置

而无需了解其是什么类型。您也可以将随机数生成器恢复为其默认设置，而无需了解这些设置是什么。

`rng` 虽然存在可能需要指定生成器类型的情况，但指定生成器类型不是必需的，因而可简化您的操作。。

如果能够避免指定生成器类型，则代码将自动适应需要使用不同生成器的各种情况，并将自动受益于新的默认随机数生成器类型中的改进属性。

旧模式和 `rng`

在低于 MATLAB 7.7 的版本中，通过直接调用带有 'seed'、'state' 或 'twister' 输入的 `rand` 或 `randn` 来控制随机数生成器的内部状态。例如，

```
rand('state',1234)
```

建议不要使用该语法，该语法会将 MATLAB 切换到“旧随机数模式”，这种情况下 `rand` 和 `randn` 将使用单独和过期的生成器，就像低于 MATLAB 7.7 的版本中的行为一样。如果可能，应更新使用旧语法的所有现有代码，以便改用 `rng`。要执行该操作，可能需要考虑一下确定使用旧代码的真实意图；有关建议和示例，请参阅用户指南中的“更换不推荐的 `rand` 和 `randn` 语法”（第 3-26 页）。

如果您或所运行的某些代码执行了将 MATLAB 置于旧模式的命令（如 `rand('state',1234)`），则可以使用

```
rng default
```

来转义旧模式并恢复为默认启动生成器。如果存在您无法修改或不允许您修改的代码，则可以使用以下命令保护该旧代码：

```
s = rng; % save current settings of the generator
% call code using legacy random number generator syntaxes
rng(s) % restore previous settings of the generator
```

以确保没有其他代码使用旧随机函数生成器。

`rng` 和 `RandStream`

`rng` 提供了一种便捷的方式来控制 MATLAB 中的随机数生成，以满足大多数常见的需求。但是，涉及多个随机数流和并行随机数生成的更复杂情况需要更复杂的工具。`RandStream` 类即是满足该需要的工具，

它提供了最强大的方式来控制随机数生成。这两个工具相互补充，其中 `rng` 基于 `RandStream` 的灵活性而构建，可提供更简单和更精炼的语法。

稀疏矩阵

- “稀疏矩阵的计算优点” (第 4-2 页)
- “构造稀疏矩阵” (第 4-3 页)
- “访问稀疏矩阵” (第 4-7 页)
- “稀疏矩阵运算” (第 4-13 页)
- “有限差分拉普拉斯算子” (第 4-25 页)
- “稀疏矩阵的图形表示” (第 4-29 页)
- “图形和矩阵” (第 4-35 页)
- “稀疏矩阵重新排序” (第 4-41 页)
- “线性方程组的迭代方法” (第 4-53 页)

稀疏矩阵的计算优点

本节内容

“内存的管理”（第 4-2 页）

“计算效率”（第 4-2 页）

内存的管理

使用稀疏矩阵存储包含众多零值元素的数据，可以节省大量内存并加快该数据的处理速度。**sparse** 是一种属性，可以将该属性分配给由 **double** 或 **logical** 元素组成的任何二维 MATLAB 矩阵。

通过 **sparse** 属性，MATLAB 可以：

- 仅存储矩阵中的非零元素及其索引。
- 不必对零元素执行运算，从而减少计算时间。

对于满矩阵，MATLAB 将在内部存储每个矩阵元素。零值元素与任何其他矩阵元素需要的存储空间量相同。但是，对于稀疏矩阵，MATLAB 只会存储非零元素及其索引。对于零值元素百分比很高的大型矩阵，此方案可以极大地减少存储数据所需的内存量。

whos 命令提供有关矩阵存储的高级信息，包括大小和存储类。例如，以下的 **whos** 列表显示了有关同一矩阵的稀疏版本和完全版本的信息。

```
M_full = magic(1100);      % Create 1100-by-1100 matrix.
M_full(M_full > 50) = 0;    % Set elements >50 to zero.
M_sparse = sparse(M_full); % Create sparse matrix of same.
```

whos

Name	Size	Bytes	Class	Attributes
M_full	1100x1100	9680000	double	
M_sparse	1100x1100	9608	double	sparse

请注意，稀疏矩阵中使用的字节数较少，因为零值元素未被存储。

计算效率

在计算效率方面，稀疏矩阵也具有显著的优点。与满矩阵的运算不同，稀疏矩阵的运算不会执行不必要的低级算术操作，例如加零 ($x+0$ 始终为 x)。这样便可大大缩短处理大量稀疏数据的程序的执行时间。

另请参阅

详细信息

- “稀疏矩阵运算”（第 4-13 页）

构造稀疏矩阵

本节内容

- “创建稀疏矩阵” (第 4-3 页)
- “导入稀疏矩阵” (第 4-6 页)

创建稀疏矩阵

MATLAB 从不会自动创建稀疏矩阵。相反，还必须确定矩阵中是否包含足够高百分比的零元素，以便利用稀疏方法。

矩阵的密度是指非零元素数目除以矩阵元素总数。对于矩阵 M，这将为

```
nnz(M) / prod(size(M));
```

或

```
nnz(M) / numel(M);
```

密度非常低的矩阵通常很适合使用稀疏格式。

将满矩阵转换为稀疏矩阵

可以使用带有单个参数的 **sparse** 函数将满矩阵转换为稀疏存储。

例如：

```
A = [ 0  0  0  5
      0  2  0  0
      1  3  0  0
      0  0  4  0];
S = sparse(A)
```

S =

```
(3,1)    1
(2,2)    2
(3,2)    3
(4,3)    4
(1,4)    5
```

列显输出中列出了 S 的非零元素及其行索引和列索引。这些元素按列排序，反映了内部数据结构体。

如果矩阵阶数不太高，可以使用 **full** 函数将稀疏矩阵转换为满存储。例如，**A = full(S)** 可反向转换该示例。

将满矩阵转换为稀疏存储并非生成稀疏矩阵的最常用方法。如果矩阵的阶数足够低可以进行满存储，则转换为稀疏存储很难显著节省内存。

直接创建稀疏矩阵

可以使用带有五个参数的 **sparse** 函数，基于一列非零元素来创建稀疏矩阵。

```
S = sparse(i,j,s,m,n)
```

i 和 **j** 分别是矩阵中非零元素的行索引和列索引的向量。**s** 是由对应的 **(i,j)** 对指定索引的非零值的向量。**m** 是生成的矩阵的行维度，**n** 是其列维度。

前一示例中的矩阵 **S** 可以直接通过以下表达式生成

```
S = sparse([3 2 3 4 1],[1 2 2 3 4],[1 2 3 4 5],4,4)
```

S =

(3,1)	1
(2,2)	2
(3,2)	3
(4,3)	4
(1,4)	5

sparse 命令具有许多备用形式。上面示例使用的形式将矩阵中的最大非零元素数设置为 **length(s)**。如果需要，可以追加第六个参数用来指定更大的最大数，这样能在以后添加非零元素，而不必重新分配稀疏矩阵。

二阶微分算子的矩阵表示形式就是一个很好的稀疏矩阵示例。它是一个三对角矩阵，其中 -2s 在对角线上，1s 在上对角线和下对角线上。有多种方式生成此类矩阵，这里只是一种可能性。

```
n = 5;
D = sparse(1:n,1:n,-2*ones(1,n),n,n);
E = sparse(2:n,1:n-1,ones(1,n-1),n,n);
S = E+D+E'
```

S =

(1,1)	-2
(2,1)	1
(1,2)	1
(2,2)	-2
(3,2)	1
(2,3)	1
(3,3)	-2
(4,3)	1
(3,4)	1
(4,4)	-2
(5,4)	1
(4,5)	1
(5,5)	-2

现在，**F = full(S)** 显示相应的满矩阵。

```
F = full(S)
```

F =

-2	1	0	0	0
1	-2	1	0	0
0	1	-2	1	0
0	0	1	-2	1
0	0	0	1	-2

基于稀疏矩阵的对角线元素创建稀疏矩阵

基于稀疏矩阵的对角线元素创建稀疏矩阵是一种常用操作，因此函数 **spdiags** 可以处理此任务。其语法是

```
S = spdiags(B,d,m,n)
```

要创建大小为 $m \times n$ 且元素在 p 对角线上的输出矩阵 S :

- B 是大小为 $\min(m,n) \times p$ 的矩阵。 B 的列是用于填充 S 对角线的值。
- d 是长度 p 的向量，其整数元素可以指定要填充的 S 对角线。

即， B 的列 j 中的元素填充 d 的元素 j 指定的对角线。

注意 如果 B 的列长度超过所替换的对角线，则上对角线从 B 列的下部获取，下对角线从 B 列的上部获取。

例如，考虑使用矩阵 B 和向量 d 。

```
B = [ 41  11   0
      52  22   0
      63  33  13
      74  44  24 ];
```

```
d = [-3
      0
      2];
```

使用这些矩阵创建 7×4 稀疏矩阵 A :

```
A = spdiags(B,d,7,4)
```

```
A =
```

(1,1)	11
(4,1)	41
(2,2)	22
(5,2)	52
(1,3)	13
(3,3)	33
(6,3)	63
(2,4)	24
(4,4)	44
(7,4)	74

在其满矩阵形式中， A 类似于：

```
full(A)
```

```
ans =
```

11	0	13	0
0	22	0	24
0	0	33	0
41	0	0	44
0	52	0	0
0	0	63	0
0	0	0	74

spdiags 还可以从稀疏矩阵中提取对角线元素，或将矩阵对角线元素替换为新值。键入 **help spdiags** 以了解详细信息。

导入稀疏矩阵

可以在 MATLAB 环境外部通过计算导入稀疏矩阵。结合使用 `spconvert` 函数与 `load` 命令导入包含索引和非零元素列表的文本文件。例如，考虑使用三列文本文件 `T.dat`，它的第一列是行索引列表，第二列是列索引列表，第三列是非零值列表。这些语句将 `T.dat` 加载到 MATLAB 中并将其转换为稀疏矩阵 `S`：

```
load T.dat  
S = spconvert(T)
```

`save` 和 `load` 命令还可以处理作为 MAT 文件中的二进制数据存储的稀疏矩阵。

另请参阅

`sparse` | `spconvert`

详细信息

- “稀疏矩阵运算” (第 4-13 页)

访问稀疏矩阵

本节内容

- “非零元素”（第 4-7 页）
- “索引和值”（第 4-8 页）
- “稀疏矩阵运算中的索引”（第 4-8 页）
- “可视化稀疏矩阵”（第 4-11 页）

非零元素

以下多条命令可以提供有关稀疏矩阵的非零元素的概要信息：

- **nnz** 返回稀疏矩阵中的非零元素数。
- **nonzeros** 返回包含稀疏矩阵的所有非零元素的列向量。
- **nzmax** 返回为稀疏矩阵的非零项分配的存储空间量。

要尝试上述中的一些命令，请加载提供的稀疏矩阵 **west0479**，该矩阵是 Harwell-Boeing 集合之一。

```
load west0479
whos
```

Name	Size	Bytes	Class	Attributes
west0479	479x479	34032	double	sparse

该矩阵为八个阶段的化工蒸馏列建模。

尝试以下命令。

```
nnz(west0479)
```

```
ans =
```

```
1887
```

```
format short e
west0479
```

```
west0479 =
```

```
(25,1)    1.0000e+00
(31,1)   -3.7648e-02
(87,1)   -3.4424e-01
(26,2)    1.0000e+00
(31,2)   -2.4523e-02
(88,2)   -3.7371e-01
(27,3)    1.0000e+00
(31,3)   -3.6613e-02
(89,3)   -8.3694e-01
(28,4)    1.3000e+02
.
.
.
```

```
nonzeros(west0479)
```

```
ans =
1.0000e+00
-3.7648e-02
-3.4424e-01
1.0000e+00
-2.4523e-02
-3.7371e-01
1.0000e+00
-3.6613e-02
-8.3694e-01
1.3000e+02
.
.
.
```

注意 使用 **Ctrl+C** 随时停止列出 **nonzeros**.

请注意，最初在默认情况下，**nnz** 与 **nzmax** 的值相同。即，非零元素数等于为非零元素分配的存储位置数。但是，如果将其他的数组元素置零，MATLAB 不会动态释放内存。将某些矩阵元素的值更改为零时会更改 **nnz** 的值，但不会更改 **nzmax** 的值。

但是，可以根据需要将尽可能多的非零元素添加到矩阵中。不受 **nzmax** 原始值的限制。

索引和值

对于任何矩阵，无论是满矩阵还是稀疏矩阵，**find** 函数都会返回非零元素的索引和值。其语法是

```
[i,j,s] = find(S);
```

find 返回向量 **i** 中的非零值的行索引、向量 **j** 中的列索引以及向量 **s** 中的自身非零值。下面的示例使用 **find** 查找稀疏矩阵中的非零索引和值。**sparse** 函数同时使用 **find** 输出和矩阵大小重新创建矩阵。

```
S1 = west0479;
[i,j,s] = find(S1);
[m,n] = size(S1);
S2 = sparse(i,j,s,m,n);
```

稀疏矩阵运算中的索引

由于稀疏矩阵是以压缩稀疏列格式存储的，因此为稀疏矩阵进行索引的相关成本与为满矩阵进行索引的相关成本不同。在只需更改稀疏矩阵中的若干元素时，这类成本可忽略不计，因此，在这类情况下，通常使用常规数组索引来重新分配值：

```
B = speye(4);
[i,j,s] = find(B);
[i,j,s]
```

```
ans =
```

```
1   1   1
2   2   1
```

```

3   3   1
4   4   1

B(3,1) = 42;
[i,j,s] = find(B);
[i,j,s]

```

ans =

```

1   1   1
3   1   42
2   2   1
3   3   1
4   4   1

```

在存储新矩阵时，为使 42 位于 (3,1) 位置，MATLAB 会在非零值向量和下标向量中插入额外的一行，然后移动 (3,1) 后面的所有矩阵值。

如果线性索引超过 $2^{48}-1$ （即当前矩阵中允许的元素数上限），使用线性索引在大型稀疏矩阵中访问或指定元素将失败。

```
S = spalloc(2^30,2^30,2);
S(end) = 1
```

Maximum variable size allowed by the program is exceeded.

要访问其线性索引大于 `intmax` 的元素，请使用数组索引：

```
S(2^30,2^30) = 1
```

S =

```
(1073741824,1073741824)      1
```

尽管在稀疏矩阵中进行索引以更改单个元素的成本可忽略不计，但该成本在循环环境下会增加，而且在大型矩阵中该操作可能会使执行速度变得很慢。因此，在需要更改大量稀疏矩阵元素的情况下，最好使用向量化方法而不要使用循环方法来执行该操作。例如，考虑稀疏单位矩阵：

```
n = 10000;
A = 4*speye(n);
```

以循环方式更改 A 的元素慢于类似的向量化运算：

```

tic
A(1:n-1,n) = -1;
A(n,1:n-1) = -1;
toc

```

Elapsed time is 0.003344 seconds.

```

tic
for k = 1:n-1
    C(k,n) = -1;
    C(n,k) = -1;
end
toc

```

Elapsed time is 0.448069 seconds.

由于 MATLAB 以压缩稀疏列格式存储稀疏矩阵，因此，在循环的每个遍历期间，它都需要移动 A 中的多个条目。

如果为稀疏矩阵预分配内存，然后以类似的方式逐个元素的方式填充，会使对稀疏数组进行索引产生大量开销：

```
S1 = spalloc(1000,1000,100000);
tic;
for n = 1:100000
    i = ceil(1000*rand(1,1));
    j = ceil(1000*rand(1,1));
    S1(i,j) = rand(1,1);
end
toc
```

Elapsed time is 2.577527 seconds.

构建索引和值向量则无需为稀疏数组进行索引，因此这种方法的速度快得多：

```
i = ceil(1000*rand(100000,1));
j = ceil(1000*rand(100000,1));
v = zeros(size(i));
for n = 1:100000
    v(n) = rand(1,1);
end

tic;
S2 = sparse(i,j,v,1000,1000);
toc
```

Elapsed time is 0.017676 seconds.

因此，最好使用构造函数（例如 `sparse` 或 `spdiags` 函数）一次构造所有稀疏矩阵。

例如，假定需要稀疏形式的坐标矩阵 C：

$$C = \begin{pmatrix} 4 & 0 & 0 & 0 & -1 \\ 0 & 4 & 0 & 0 & -1 \\ 0 & 0 & 4 & 0 & -1 \\ 0 & 0 & 0 & 4 & -1 \\ 1 & 1 & 1 & 1 & 4 \end{pmatrix}$$

使用 `sparse` 函数，以及行下标、列下标和值组成的三联对组，直接构造该五列矩阵：

```
i = [1 5 2 5 3 5 4 5 1 2 3 4 5]';
j = [1 1 2 2 3 3 4 4 5 5 5 5 5]';
s = [4 1 4 1 4 1 4 1 -1 -1 -1 -1 4]';
C = sparse(i,j,s)
```

C =

(1,1)	4
(5,1)	1
(2,2)	4
(5,2)	1
(3,3)	4
(5,3)	1

(4,4)	4
(5,4)	1
(1,5)	-1
(2,5)	-1
(3,5)	-1
(4,5)	-1
(5,5)	4

输出中值的顺序反映了底层的按列存储。有关 MATLAB 如何存储稀疏矩阵的详细信息，请参阅 John R. Gilbert、Cleve Moler 和 Robert Schreiber 合著的 Sparse Matrices In Matlab:Design and Implementation, (SIAM Journal on Matrix Analysis and Applications, 13:1, 333–356 (1992))。

可视化稀疏矩阵

以图的形式查看非零元素在稀疏矩阵内的分布通常很有用。MATLAB `spy` 函数生成稀疏结构的模板视图，其中图表中的每点代表一个非零数组元素的位置。

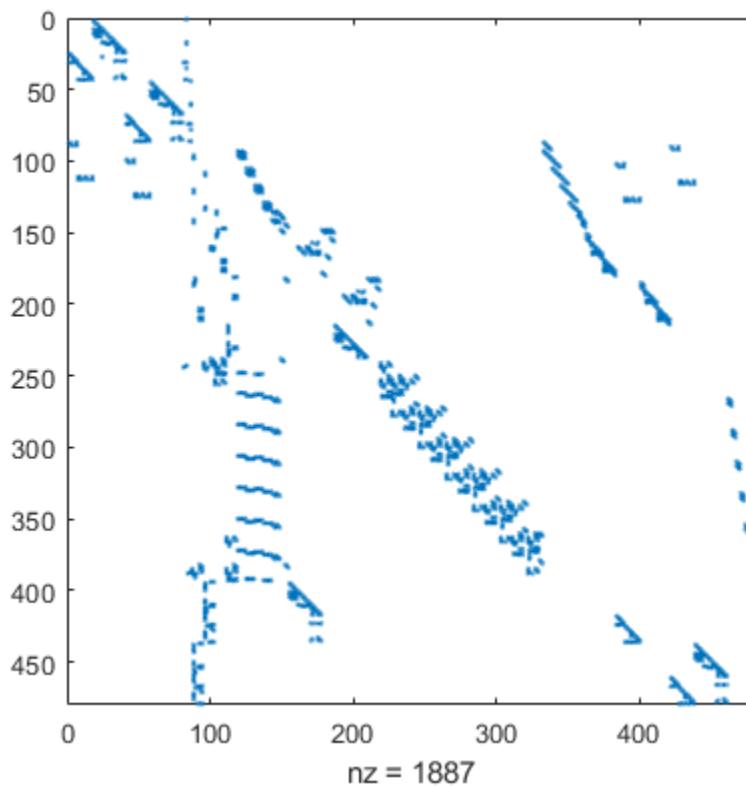
例如：

加载提供的稀疏矩阵 `west0479`，该矩阵是 Harwell-Boeing 集合之一。

```
load west0479
```

查看稀疏结构体。

```
spy(west0479)
```



另请参阅

`sparse`

详细信息

- “稀疏矩阵的计算优点”（第 4-2 页）
- “构造稀疏矩阵”（第 4-3 页）
- “稀疏矩阵运算”（第 4-13 页）

稀疏矩阵运算

运算效率

计算复杂度

稀疏运算的计算复杂度与矩阵中的非零元素数 `nnz` 成比例。计算复杂度在线性上还与矩阵的行大小 `m` 和列大小 `n` 相关，但与积 `m*n` (零元素和非零元素总数) 无关。

相当复杂的运算（例如对稀疏线性方程求解）的复杂度涉及如排序和填充之类的因素，已在上一部分中对这些因素进行了讨论。通常，稀疏矩阵运算所需的计算时间通常与非零元素数量中的算术运算数成比例。

算法详细信息

根据以下规则，稀疏矩阵通过计算传播：

- 接受矩阵并返回标量或等尺寸向量的函数始终都以满存储格式生成输出。例如，无论其输入是完全还是稀疏模式，`size` 函数始终都返回满向量。
- 接受标量或向量并返回矩阵的函数始终都返回完全结果，例如 `zeros`、`ones`、`rand` 和 `eye`。该操作十分必要，可以避免意外引入稀疏。`zeros(m,n)` 的稀疏模拟函数即是 `sparse(m,n)`。`rand` 和 `eye` 的稀疏模拟函数分别是 `sprand` 和 `speye`。函数 `ones` 没有稀疏模拟。
- 接受矩阵并返回矩阵或向量的一元函数保留存储类的操作数。如果 `S` 是稀疏矩阵，则 `chol(S)` 也是稀疏矩阵，`diag(S)` 是稀疏向量。如 `max` 和 `sum` 之类的列级函数也返回稀疏向量，即使这些向量完全为非零元素也是如此。该规则的重要例外是 `sparse` 和 `full` 函数。
- 如果两个操作数都是稀疏元素，则二元运算符生成稀疏结果；如果两个操作数都是完全元素，则生成完全结果。对于混合操作数，除非运算保留稀疏性，否则生成完全结果。如果 `S` 是稀疏矩阵，而 `F` 是满矩阵，则 `S+F`、`S*F` 和 `F\S` 是满矩阵，而 `S.*F` 和 `S&F` 是稀疏矩阵。在某些情况下，即使矩阵包含很少的零元素，也可能是稀疏结果。
- 使用 `cat` 函数或方括号串联矩阵会为混合操作数生成稀疏结果。

置换与重新排序

可以通过以下两种方式表示稀疏矩阵 `S` 的行和列置换：

- 置换矩阵 `P` 对作为 `P*S` 的 `S` 有效，或对作为 `S*P'` 的列有效。
- 置换向量 `p` 是包含 `1:n` 置换的满向量。对作为 `S(p,:)` 的 `S` 行有效，或对作为 `S(:,p)` 的列有效。

例如：

```
p = [1 3 4 2 5]
I = eye(5,5);
P = I(p,:);
e = ones(4,1);
S = diag(11:11:55) + diag(e,1) + diag(e,-1)
```

`p =`

1 3 4 2 5

`P =`

```
1  0  0  0  0  
0  0  1  0  0  
0  0  0  1  0  
0  1  0  0  0  
0  0  0  0  1
```

S =

```
11  1  0  0  0  
1  22 1  0  0  
0  1  33 1  0  
0  0  1  44 1  
0  0  0  1  55
```

现在，可以使用置换向量 p 和置换矩阵 P 尝试一些置换。例如，语句 S(p,:) 和 P*S 返回相同的矩阵。

S(p,:)

ans =

```
11  1  0  0  0  
0  1  33 1  0  
0  0  1  44 1  
1  22 1  0  0  
0  0  0  1  55
```

P*S

ans =

```
11  1  0  0  0  
0  1  33 1  0  
0  0  1  44 1  
1  22 1  0  0  
0  0  0  1  55
```

同样，S(:,p) 和 S*p' 都生成

S*p'

ans =

```
11  0  0  1  0  
1  1  0  22 0  
0  33 1  1  0  
0  1  44 0  1  
0  0  1  0  55
```

如果 P 是稀疏矩阵，则两种表示形式都使用与 n 成比例的存储，可以在与 nnz(S) 成正比的时间向 S 应用任一种表示形式。向量表示形式略为简洁和有效，因此除了 LU (三角) 分解中的主元置换返回与完全 LU 分解兼容的矩阵外，许多稀疏矩阵置换例程都返回满行向量。

要在两种置换表示之间转换：

```
n = 5;  
I = speye(n);  
Pr = I(p,:);
```

```
Pc = I(:,p);
pc = (1:n)*Pc
```

```
pc =
```

```
1 3 4 2 5
```

```
pr = (Pr*(1:n))'
```

```
pr =
```

```
1 3 4 2 5
```

P 的逆为 **R** = **P**'。可以通过 **r(p) = 1:n** 计算 **p** 的逆。

```
r(p) = 1:5
```

```
r =
```

```
1 4 2 3 5
```

重新排序以改变稀疏性

对矩阵列重新排序通常可以使其 LU 或 QR 因子更加稀疏。对列和列重新排序通常可以使其 Cholesky 因子更加稀疏。此类最简单的重新排序是按照非零元素计数为列排序。对于具有及其不规则结构的矩阵而言，这有时是一个不错的重新排序方法，尤其是行或列的非零元素计数出现重大变化时更是如此。

colperm 函数按每列中非零元素数量从小到大的顺序重排矩阵列，计算矩阵的置换。

重新排序以减少带宽

反向 Cuthill-McKee 排序用于减少矩阵的轮廓或带宽。该排序方法不能保证会找到尽可能最小的带宽，但通常都能找到。**symrcm** 函数实际上处理的是对称矩阵 $A + A'$ 中的非零结构体。但对于非对称矩阵，该函数的结果也很有用。此种排序对源自一维问题或某种意义上细长的问题的矩阵很有用。

近似最小度排序

节点在图中的度是到该节点的连接数。这与邻接矩阵的相应行中的非对角非零元素数相同。近似最小度算法基于高斯消去法或 Cholesky 分解期间这些度的更改来进行排序。该算法很复杂并且功能强大，通常会生成比大多数其他排序稀疏的因子，包含列计数和反向 Cuthill-McKee。由于记录每个节点的度非常耗时，因此近似最小度算法使用近似度而不是精确度。

以下 MATLAB 函数可以实现近似最小度算法：

- **symamd** - 用于对称矩阵。
- **colamd** - 用于 A^*A' 或 A'^*A 形式的非对称矩阵和对称矩阵。

请参阅“稀疏矩阵的重新排序和分解”（第 4-16 页）以了解使用 **symamd** 的示例。

可以使用 **spparms** 函数更改与这些算法详细信息相关不同的参数。

有关 **colamd** 和 **symamd** 所用算法的详细信息，请参阅 [5]。这些算法使用的近似度基于 [1]。

嵌套剖分排序

dissect 函数所实现的嵌套剖分排序算法与近似最小度排序类似，即通过将矩阵视为图的邻接矩阵，对矩阵的行和列进行重新排序。该算法通过将图中的顶点对折叠在一起，大幅减小问题的规模。在对较小的图重新排序后，该算法随即通过应用投影和优化步骤，将图恢复至原始大小。

与其他重新排序方法相比，嵌套剖分算法可生成高质量的重新排序，尤其适用于有限元矩阵。有关嵌套剖分排序算法的详细信息，请参阅 [7]。

稀疏矩阵分解

LU 分解

如果 S 是稀疏矩阵，则以下命令返回三个稀疏矩阵： L 、 U 和 P ，这样 $P*S = L*U$ 。

```
[L,U,P] = lu(S);
```

lu 使用高斯消去法和部分主元消元法获取因子。置换矩阵 P 仅有 n 个非零元素。和稠密矩阵一样，语句 $[L,U] = lu(S)$ 返回置换的单位下三角矩阵和其积为 S 的上三角矩阵。**lu(S)** 本身在单个矩阵中返回 L 和 U ，没有主元信息。

有三个输出的语法 $[L,U,P] = lu(S)$ 通过数值部分主元消元法选择 P ，但不执行主元消元法以改进 LU 因子中的稀疏性。另一方面，有四个输出的语法 $[L,U,P,Q] = lu(S)$ 通过阈值部分主元消元法选择 P ，并选择 P 和 Q 以改善 LU 因子中的稀疏性。

可以使用以下项控制稀疏矩阵中的主元

```
lu(S,thresh)
```

其中 **thresh** 是 $[0,1]$ 中的主元阈值。列中对角元素的模小于该列中任何子对角元素模的 **thresh** 倍时会选主元。**thresh = 0** 强迫选择对角主元。**thresh = 1** 是默认值。（**thresh** 的默认值是四输出语法的 **0.1**）。

调用不超过三个输出的 **lu** 时，MATLAB 会在分解期间自动分配存储稀疏 L 和 U 因子所必需的内存。除了四个输出的语法外，MATLAB 不使用任何符号 LU 预分解确定内存要求和提前设置数据结构体。

稀疏矩阵的重新排序和分解

此示例说明重新排序与分解对稀疏矩阵的影响。

如果您求得可减少填充项的良好列置换矩阵 p （可能通过 **symrcm** 或 **colamd** 求得），则计算 **lu(S(:,p))** 所需的时间和存储将少于计算 **lu(S)** 所需的时间和存储。

使用布基球示例创建稀疏矩阵。

```
B = bucky;
```

B 在每一个行和列都有恰好 3 个非零元素。

分别使用 **symrcm** 和 **symamd** 创建两个置换 r 和 m 。

```
r = symrcm(B);
m = symamd(B);
```

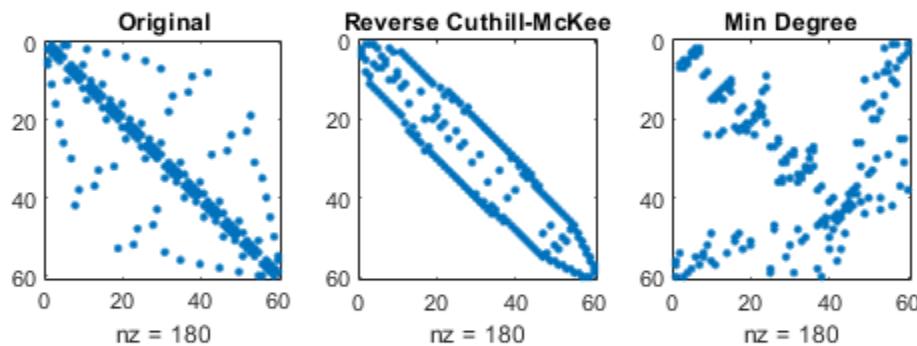
这两个置换是对称反向 Cuthill-McKee 排序和对称近似最小度排序。

创建稀疏模式图，显示具有以下三个不同数序的三个布基球图邻接矩阵。原始数序对应的局部五边形结构在其他结构中并不明显。

```
figure
subplot(1,3,1)
spy(B)
title('Original')

subplot(1,3,2)
spy(B(r,r))
title('Reverse Cuthill-McKee')

subplot(1,3,3)
spy(B(m,m))
title('Min Degree')
```



反向 Cuthill-McKee 排序 r 减小了带宽，并将所有非零元素集中在对角线附近。近似最小度排序 m 则生成了包含大块零的类分形结构。

要查看在布基球 LU 分解中生成的填充元素，请使用稀疏单位矩阵 `speye` 在 B 的对角线上插入 -3。

```
B = B - 3*speye(size(B));
```

由于每一行的和现在为零，因此新的 B 实际为奇异矩阵，但它仍有助于计算其 LU 分解。当仅使用一个输出参数调用时，`lu` 会在一个稀疏矩阵中返回两个三角矩阵 L 和 U 。该矩阵中的非零元素数量是在解算涉及 B 的线性方程组时所需的时间和存储度量。

以下是当前考虑的三个置换的非零计数。

- $\text{lu}(\mathbf{B})$ (原始) : 1022
- $\text{lu}(\mathbf{B}(r,r))$ (反向 Cuthill-McKee 排序) : 968
- $\text{lu}(\mathbf{B}(m,m))$ (近似最小度) : 636

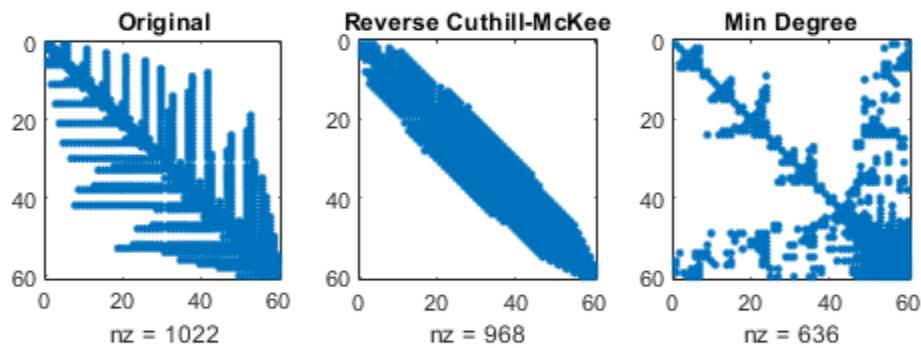
尽管这只是一个很小的示例，但结果却非常典型。原始数序方案产生的填充元素最多。反向 Cuthill-McKee 排序的填充元素集中在条带以内，但几乎与前两个排序一样广泛。对于近似最小度排序，在消元过程中保留了相对较大的零块，并且填充量远小于其他排序所生成的填充量。

以下 spy 绘图反映了每个重新排序的特征。

```
figure
subplot(1,3,1)
spy(lu(B))
title('Original')

subplot(1,3,2)
spy(lu(B(r,r)))
title('Reverse Cuthill-McKee')

subplot(1,3,3)
spy(lu(B(m,m)))
title('Min Degree')
```



Cholesky 分解

如果 S 是对称 (或 Hermitian) 的正定稀疏矩阵, 下面的语句返回上三角形稀疏矩阵 R , 这样 $R'^*R = S$ 。

R = chol(S)

chol 不会针对稀疏度自动选主元, 但可以计算近似最小度和描述文件极限置换以用于 **chol(S(p,p))**。

由于 Cholesky 算法不会针对稀疏度选主元, 也不需要选主元以保持数值稳定性, **chol** 将快速计算所需的内存量并在分解开始时分配所有的内存。可以使用 **symbfact** 计算分配的内存量, 该函数使用与 **chol** 相同的算法。

QR 分解

MATLAB 通过以下语句计算稀疏矩阵 S 的完整 QR 分解

[Q,R] = qr(S)

或

[Q,R,E] = qr(S)

但这通常不切实际。酉矩阵 Q 通常无法拥有大量的零元素。可以使用更为实际的替代方法, 该方法有时也称为 “Q-less QR 分解”。

通过一个稀疏输入参数和一个输出参数

R = qr(S)

仅返回 QR 分解的上三角形部分。矩阵 R 为与标准方程相关的矩阵提供 Cholesky 分解:

R'*R = S'*S

但是, 避免在计算 $S'*S$ 时丢失固有的数值信息。

通过两个具有相同行数的输入参数和两个输出参数, 语句

[C,R] = qr(S,B)

对 B 应用正交变换, 生成 $C = Q'^*B$, 但不计算 Q 。

Q-less QR 分解允许解决稀疏最小二乘问题

$$\text{minimize} \|Ax - b\|_2$$

通过两步

[c,R] = qr(A,b);
x = R\c

如果 A 是稀疏矩阵但不是方阵, MATLAB 对线性方程求解反斜杠运算符使用这些步长。

x = A\b

您也可以自己进行分解, 并检查 R 是否发生秩亏情况。

也可以通过不同的右端 \mathbf{b} 对一个序列的最小二乘线性方程组求解，计算 $\mathbf{R} = \text{qr}(\mathbf{A})$ 时该右端不必为已知。该方法使用以下代码对“半标准方程” $\mathbf{R}' * \mathbf{R} * \mathbf{x} = \mathbf{A}' * \mathbf{b}$ 求解

```
x = R'\(A'*b)
```

然后应用一步迭代加细以减少舍入误差：

```
r = b - A*x;
e = R'\(R'\(A'*r));
x = x + e
```

不完全分解

`ilu` 和 `ichol` 函数提供近似不完全分解，它们可用作稀疏迭代方法的预条件子。

`ilu` 函数生成三个不完全的下-上 (ILU) 分解：零填充 ILU (`ILU(0)`)、Crout 版本的 ILU (`ILUC(tau)`)，以及具有阈值调降和主元的 ILU (`ILUTP(tau)`)。`ILU(0)` 从不选主元，并且生成因子仅在输入矩阵具有非零元素的位置包含非零元素。但是，`ILUC(tau)` 和 `ILUTP(tau)` 通过用户定义的调降容差 `tau` 执行基于阈值的调降。

例如：

```
A = gallery('neumann', 1600) + speye(1600);
```

```
ans =
```

```
7840
```

```
nnz(lu(A))
```

```
ans =
```

```
126478
```

显示 \mathbf{A} 包含 7840 个非零元素，其完全 LU 分解包含 126478 个非零元素。另一方面，以下代码显示不同的 ILU 输出：

```
[L,U] = ilu(A);
nnz(L)+nnz(U)-size(A,1)
```

```
ans =
```

```
7840
```

```
norm(A-(L*U).*spones(A),'fro')./norm(A,'fro')
```

```
ans =
```

```
4.8874e-17
```

```
opts.type = 'ilutp';
opts.droptol = 1e-4;
[L,U,P] = ilu(A, opts);
nnz(L)+nnz(U)-size(A,1)
```

```
ans =
```

```
31147
```

```
norm(P*A - L*U,'fro')./norm(A,'fro')
```

```

ans =
9.9224e-05

opts.type = 'crout';
[L,U,P] = ilu(A, opts);
nnz(L)+nnz(U)-size(A,1)

ans =
31083

norm(P*A-L*U,'fro')./norm(A,'fro')

ans =
9.7344e-05

```

这些计算显示零填充因子包含 7840 个非零元素, **ILUTP(1e-4)** 因子包含 31147 个非零元素, **ILUC(1e-4)** 因子包含 31083 个非零元素。另外, 零填充因子积的相对误差在 A 的模式中实质上为零。最后, 通过阈值调降生成的分解中的相对误差类似于调降容差, 虽然不能保证一定会发生此情况。请参阅 **ilu** 参考页以了解更多选项和详细信息。

ichol 函数提供对称正定稀疏矩阵的零填充不完全 Cholesky 分解 (**IC(0)**) 以及基于阈值的调降不完全 Cholesky 分解 (**ICT(tau)**)。这些分解是上述不完全 LU 分解的模拟, 具有许多相同特点。例如:

```

A = delsq(numgrid('S',200));
nnz(A)

ans =

```

```

195228

nnz(chol(A,'lower'))

ans =

```

7762589

显示 A 包含 195228 个非零元素, 其没有重新排序的完全 Cholesky 分解包含 7762589 个非零元素。相比之下:

```

L = ichol(A);
nnz(L)

ans =

```

```

117216

norm(A-(L*L').*spones(A),'fro')./norm(A,'fro')

ans =

```

```

3.5805e-17

opts.type = 'ict';
opts.droptol = 1e-4;
L = ichol(A,opts);
nnz(L)

```

```

ans =
1166754
norm(A-L*L','fro')./norm(A,'fro')
ans =
2.3997e-04

```

IC(0) 仅在 A 的下三角模式以及 A 的匹配因子积模式中拥有非零模式。另外，**ICT(1e-4)** 因子远远比完全 Cholesky 分解稀疏，并且 A 与 L*L' 之间的相对误差类似于调降容差。值得注意的是，与 **chol** 提供的因子不同，**ichol** 提供的默认因子是下三角。有关详细信息，请参阅 **ichol** 参考页。

特征值和奇异值

可以使用两个函数计算一些指定的特征值或奇异值。**svds** 基于 **eigs**。

用于计算一些特征值或奇异值的函数

函数	说明
eigs	少数特征值
svds	少数奇异值

这些函数最常用于稀疏矩阵，但它们可以用于满矩阵，甚至是 MATLAB 代码中定义的线性运算函数。

语句

```
[V,lambda] = eigs(A,k,sigma)
```

查找矩阵 A 最靠近 “shift” **sigma** 的 k 特征值及相应的特征向量。如果忽略 **sigma**，会找到模最大的特征值。如果 **sigma** 为零，会找到模最小的特征值。可以包含第二个矩阵 B 以避免广义特征值问题：Au = λBu。

语句

```
[U,S,V] = svds(A,k)
```

查找 A 的 k 个最大奇异值，而

```
[U,S,V] = svds(A,k,'smallest')
```

查找 k 个最小奇异值。

eigs 和 **svds** 中使用的数值方法在 [6] 中进行了介绍。

稀疏矩阵的最小特征值

此示例说明如何求稀疏矩阵的最小特征值和特征向量。

在 L 形二维域中的 65×65 网格中设置五点拉普拉斯微分算子。

```

L = numgrid('L',65);
A = delsq(L);

```

确定非零元素的阶数和数量。

```
size(A)
```

```
ans = 1×2
```

```
2945      2945
```

```
nnz(A)
```

```
ans = 14473
```

A 是阶数为 2945 且包含 14,473 个非零元素的矩阵。

计算最小特征值和特征向量。

```
[v,d] = eigs(A,1,'smallests');
```

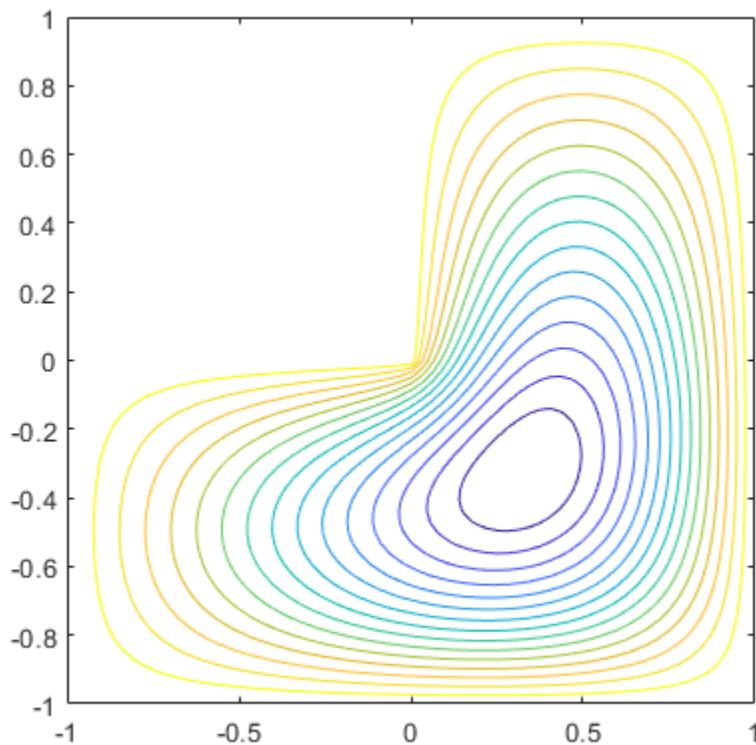
在相应的网格点上分布特征向量的分量，并生成结果的等高线图。

```
L(L>0) = full(v(L(L>0)));
```

```
x = -1:1/32:1;
```

```
contour(x,x,L,15)
```

```
axis square
```



参考

- [1] Amestoy, P. R., T. A. Davis, and I. S. Duff, "An Approximate Minimum Degree Ordering Algorithm," *SIAM Journal on Matrix Analysis and Applications*, Vol. 17, No. 4, Oct. 1996, pp. 886-905.
- [2] Barrett, R., M. Berry, T. F. Chan, et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1994.
- [3] Davis, T.A., Gilbert, J. R., Larimore, S.I., Ng, E., Peyton, B., "A Column Approximate Minimum Degree Ordering Algorithm," *Proc. SIAM Conference on Applied Linear Algebra*, Oct. 1997, p. 29.
- [4] Gilbert, John R., Cleve Moler, and Robert Schreiber, "Sparse Matrices in MATLAB: Design and Implementation," *SIAM J. Matrix Anal. Appl.*, Vol. 13, No. 1, January 1992, pp. 333-356.
- [5] Larimore, S. I., *An Approximate Minimum Degree Column Ordering Algorithm*, MS Thesis, Dept. of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, 1998.
- [6] Saad, Yousef, *Iterative Methods for Sparse Linear Equations*. PWS Publishing Company, 1996.
- [7] Karypis, George and Vipin Kumar. "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs." *SIAM Journal on Scientific Computing*. Vol. 20, Number 1, 1999, pp. 359-392.

另请参阅

详细信息

- "稀疏矩阵的计算优点" (第 4-2 页)
- "构造稀疏矩阵" (第 4-3 页)
- "访问稀疏矩阵" (第 4-7 页)

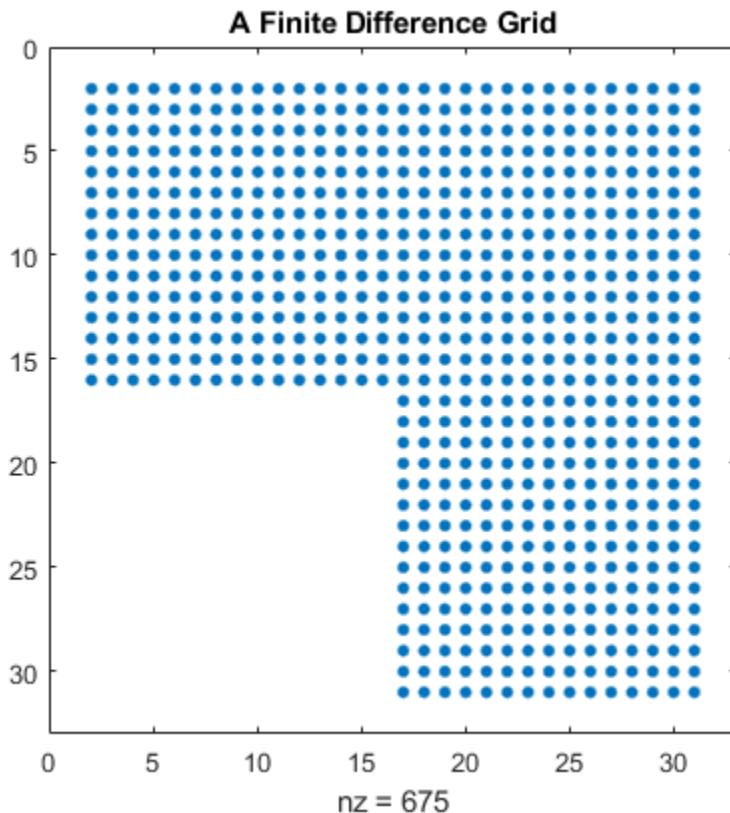
有限差分拉普拉斯算子

此示例说明如何在 L 形域中计算和表示有限差分拉普拉斯算子。

域

`numgrid` 函数为 L 形域内的点编号。`spy` 函数可用于可视化矩阵中非零元素的模式。使用这两个函数生成并显示 L 形域。

```
n = 32;
R = 'L';
G = numgrid(R,n);
spy(G)
title('A Finite Difference Grid')
```



取一个较小版本的矩阵作为样例。

```
g = numgrid(R,10)
```

```
g = 10×10
```

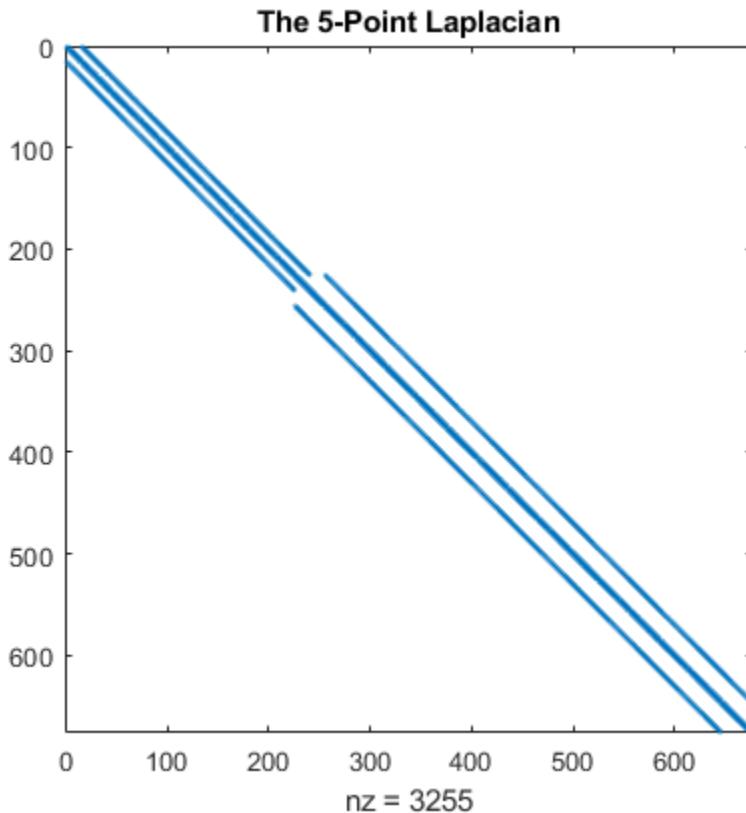
0	0	0	0	0	0	0	0	0	0
0	1	5	9	13	17	25	33	41	0
0	2	6	10	14	18	26	34	42	0
0	3	7	11	15	19	27	35	43	0
0	4	8	12	16	20	28	36	44	0
0	0	0	0	0	21	29	37	45	0

```
0  0  0  0  0  22  30  38  46  0  
0  0  0  0  0  23  31  39  47  0  
0  0  0  0  0  24  32  40  48  0  
0  0  0  0  0  0  0  0  0  0
```

离散拉普拉斯算子

使用 `delsq` 生成离散拉普拉斯算子。同样使用 `spy` 函数显示矩阵元素的图形效果。

```
D = delsq(G);  
spy(D)  
title('The 5-Point Laplacian')
```



确定内部点的数量。

```
N = sum(G(:)>0)
```

```
N = 675
```

Dirichlet 边界值问题

求解稀疏线性方程组的 Dirichlet 边界值问题。问题设置：

`delsq(u) = 1` 在内部，`u = 0` 在边界上。

```
rhs = ones(N,1);  
if (R == 'N') % For nested dissection, turn off minimum degree ordering.
```

```

spparms('autommd',0)
u = D\rhs;
spparms('autommd',1)
else
    u = D\rhs; % This is used for R=='L' as in this example
end

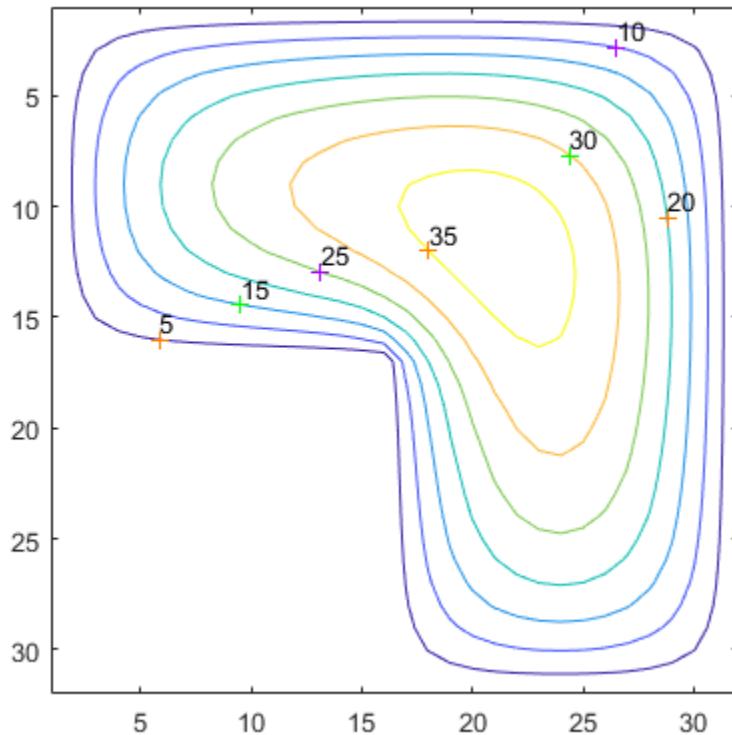
```

将解映射到 L 形网格并绘制等高线图。

```

U = G;
U(G>0) = full(u(G>0)));
clabel(contour(U));
prism
axis square ij

```

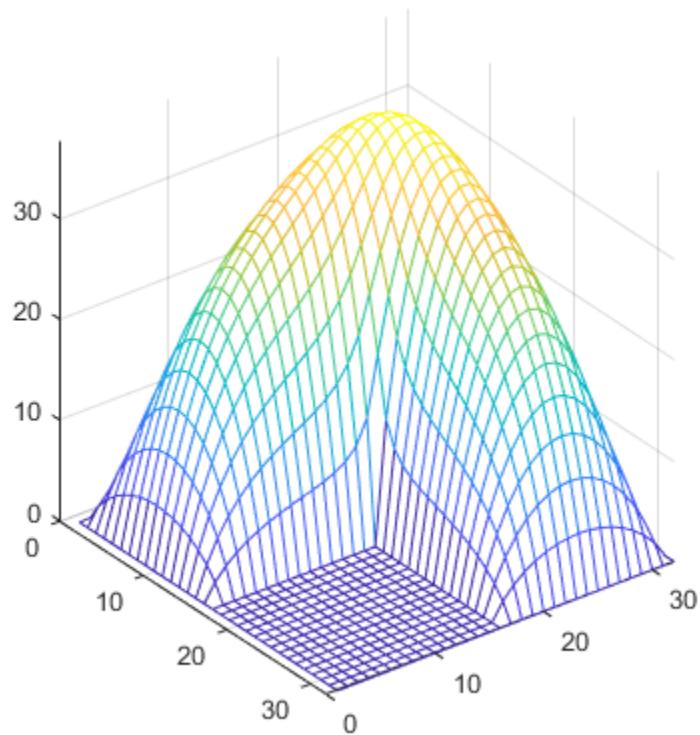


现在，以网格图形形式呈现该解。

```

mesh(U)
axis([0 n 0 n 0 max(max(U))])
axis square ij

```



另请参阅

`spy`

稀疏矩阵的图形表示

此示例说明 NASA 翼型的有限元网格，包括两个尾翼。有关翼型发展历史的详细信息，请访问 NACA Airfoils (nasa.gov)。

数据存储在文件 `airfoil.mat` 中。数据由 4253 对 (x,y) 网格点坐标组成。此外，还包含一个由 12,289 对索引 (i,j) 组成的数组，这些索引指定网格点之间的连接。

将数据文件加载到工作区。

```
load airfoil
```

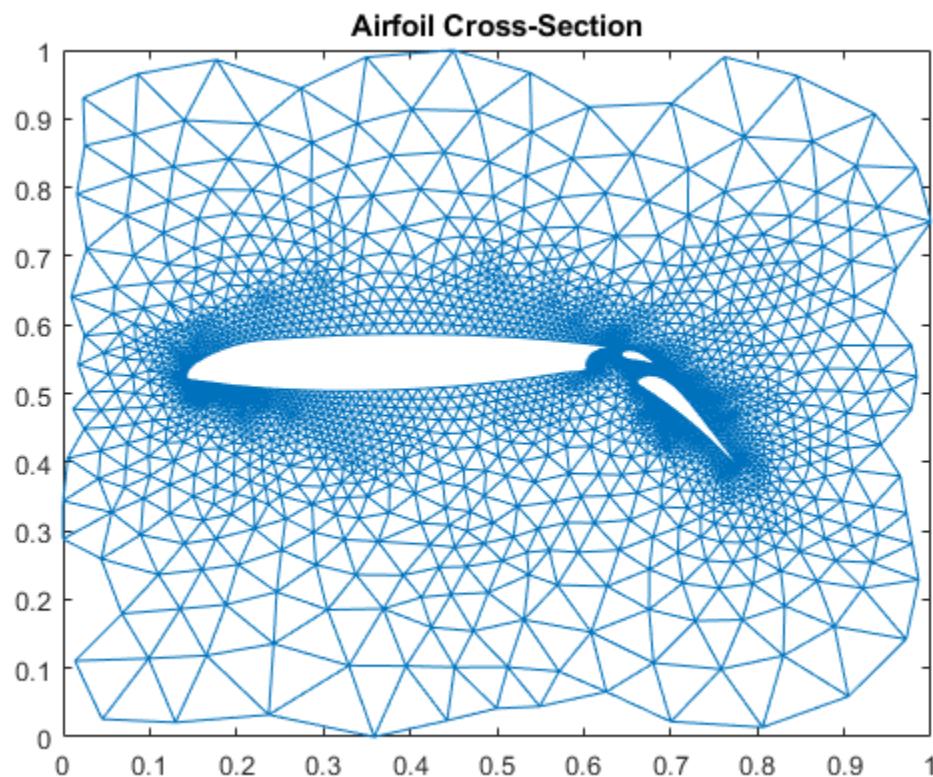
查看有限元网格

首先，将 x 和 y 分别以 2^{-32} 为比例缩小，使其处于范围 $[0, 1]$ 内。然后，从 (i,j) 连接点构造稀疏邻接矩阵，并将其设为正定矩阵。最后，使用 (x,y) 作为顶点（网格点）的坐标绘制邻接矩阵。

```
% Scaling x and y
x = pow2(x,-32);
y = pow2(y,-32);

% Forming the sparse adjacency matrix and making it positive definite
n = max(max(i),max(j));
A = sparse(i,j,-1,n,n);
A = A + A';
d = abs(sum(A)) + 1;
A = A + diag(sparse(d));

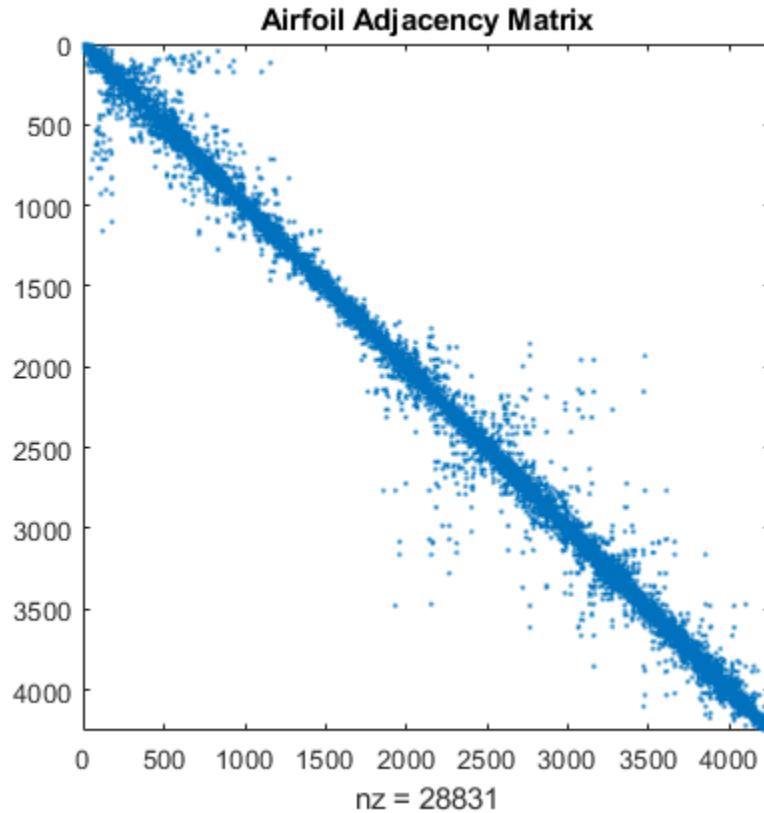
% Plotting the finite element mesh
gplot(A,[x y])
title('Airfoil Cross-Section')
```



可视化稀疏模式

您可以使用 `spy` 可视化矩阵中的非零元素，该函数尤其适用于查看稀疏矩阵中的稀疏模式。`spy(A)` 可以绘制矩阵 A 的稀疏模式。

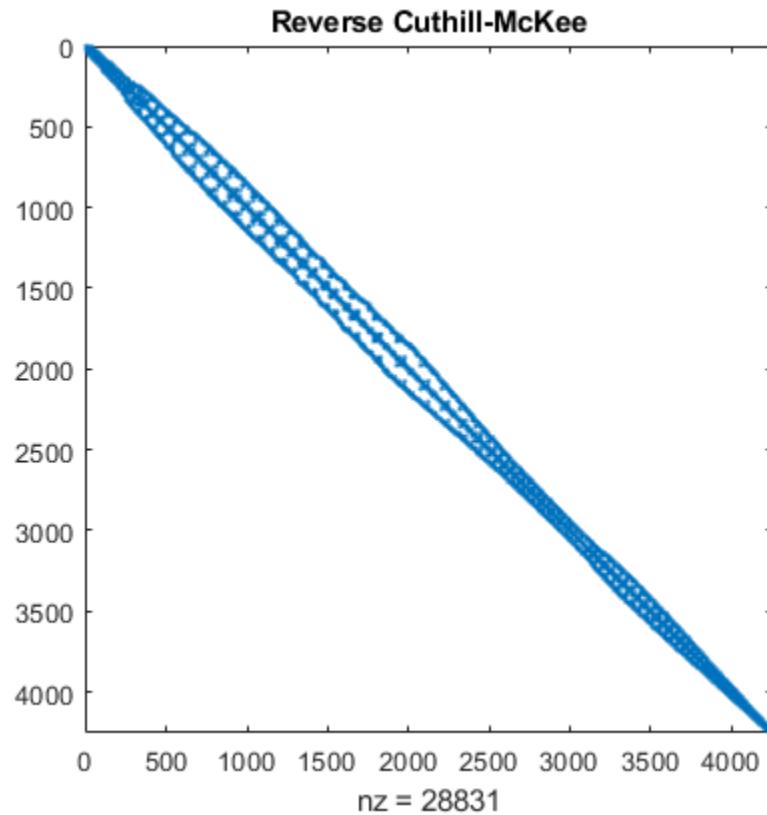
```
spy(A)
title('Airfoil Adjacency Matrix')
```



对称重新排序 - 反向 Cuthill-McKee

`symrcm` 使用反向 Cuthill-McKee 方法对邻接矩阵重新排序。`r = symrcm(A)` 返回置换向量 `r`，因此，相对于 `A`，`A(r,r)` 往往具有更接近对角线的对角线元素。对于来自“瘦长”问题的矩阵的 LU 或 Cholesky 分解来说，这是一种很好的预排序方法。此方法对于对称和非对称矩阵都适用。

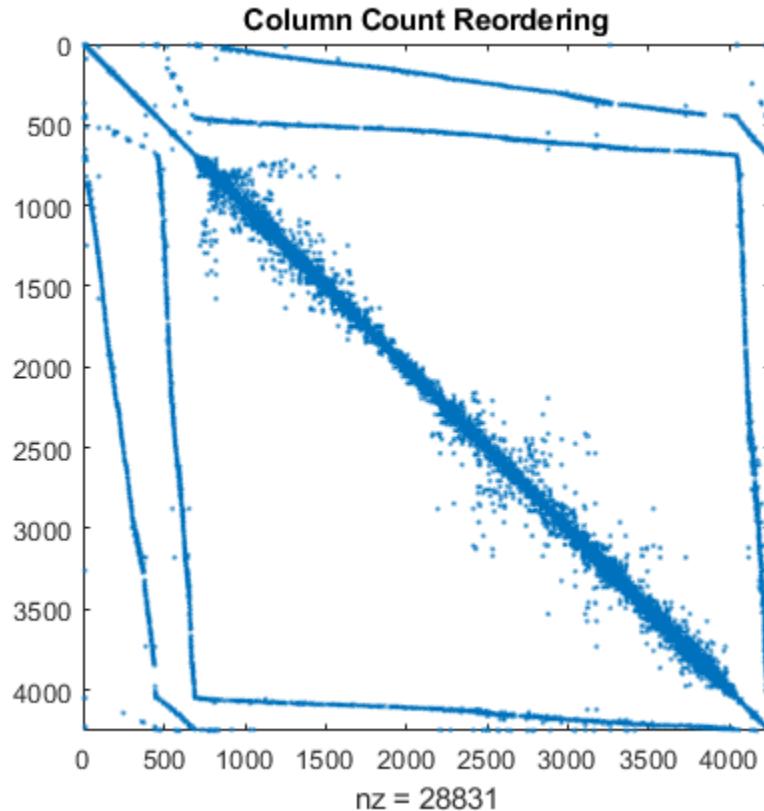
```
r = symrcm(A);
spy(A(r,r))
title('Reverse Cuthill-McKee')
```



对称重新排序 - 列置换

使用 `j = COLPERM(A)` 可返回一个置换向量，该向量以非零计数的非递减顺序对稀疏矩阵 A 的各列进行重新排序。此方法有时很有用，可作为对 LU 分解的预排序方法，如 `lu(A(:,j))`。

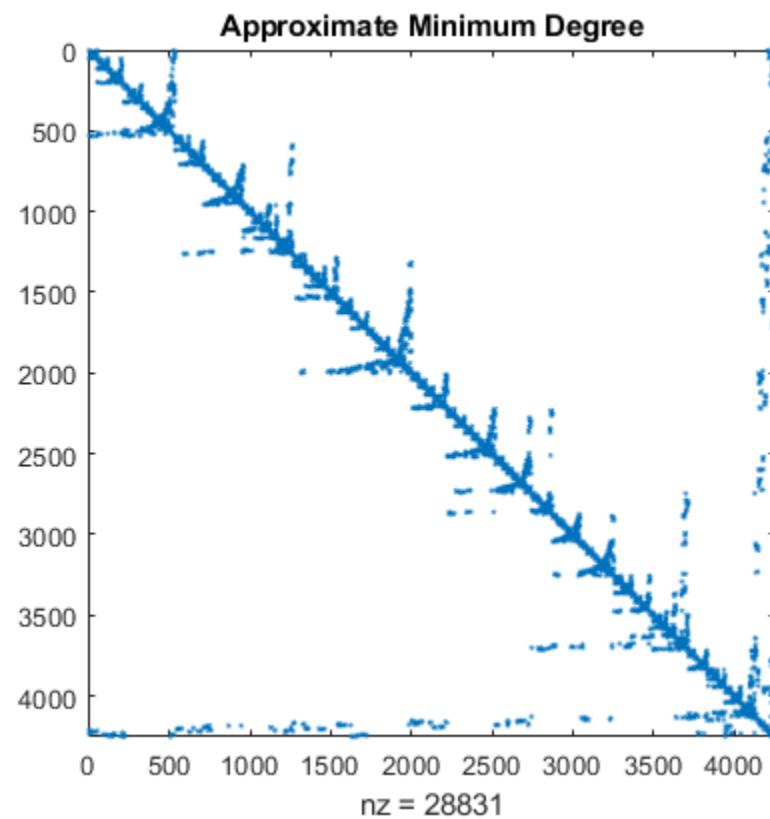
```
j = colperm(A);
spy(A(j,j))
title('Column Count Reordering')
```



对称重新排序 - 对称近似最小度

`symamd` 可以进行对称近似最小度置换。对于对称正定矩阵 A , 命令 `p = symamd(S)` 返回置换向量 p , 因此 $S(p,p)$ 倾向于比 S 具有更稀疏的 Cholesky 因子。有时 `symamd` 也适用于对称的非正定矩阵。

```
m = symamd(A);
spy(A(m,m))
title('Approximate Minimum Degree')
```



另请参阅

[colperm](#) | [spy](#) | [symamd](#) | [symrcm](#)

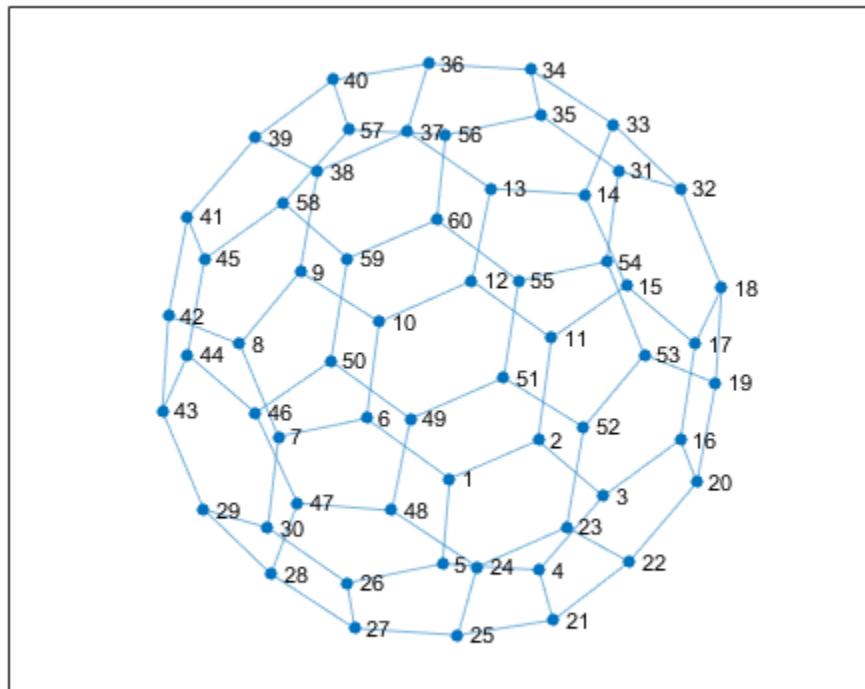
图形和矩阵

此示例说明稀疏矩阵的应用并解释了图形与矩阵之间的关系。

图形是一组相互之间具有指定连接或边的节点。图形有许多形状和大小。Buckminster Fuller 多面穹顶（也就是足球或碳 60 分子的形状）的连接图就是这样一个示例。

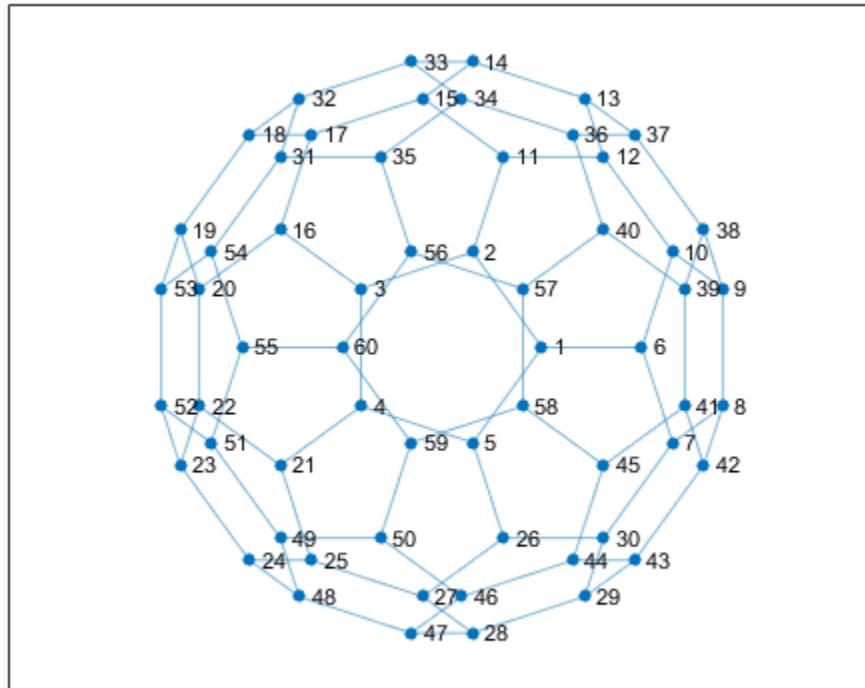
在 MATLAB® 中，您可以使用 **bucky** 函数来生成多面穹顶的图形。

```
[B,V] = bucky;  
G = graph(B);  
p = plot(G);  
axis equal
```



此外，也可以指定节点的坐标，以更改图形的显示。

```
p.XData = V(:,1);  
p.YData = V(:,2);
```



bucky 函数可用于创建图形，因为它返回邻接矩阵。邻接矩阵是一种表示图形中的节点和边的方式。

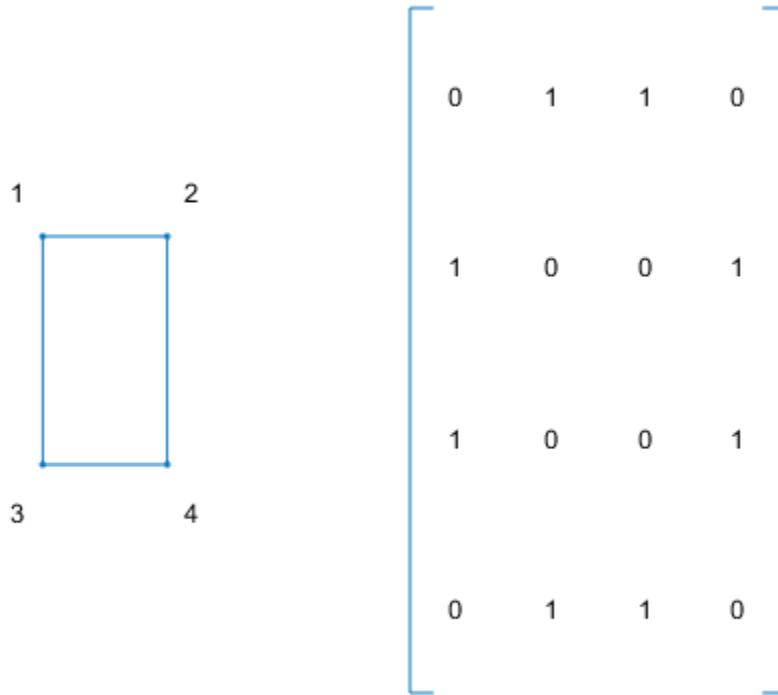
要构造图形的邻接矩阵，节点应按 1 至 N 进行编号。如果节点 i 连接至节点 j，则 $N \times N$ 矩阵的每个元素 (i,j) 都设置为 1，否则设置为 0。因此，对于无向图，邻接矩阵是对称的，但有向图不必如此。

例如，下面展示了一个简单的图形及其关联的邻接矩阵。

```
% Define a matrix A.
A = [0 1 1 0 ; 1 0 0 1 ; 1 0 0 1 ; 0 1 1 0];

% Draw a picture showing the connected nodes.
cla
subplot(1,2,1);
gplot(A,[0 1;1 0;0 1;1 0],'.-');
text([-0.2, 1.2 -0.2, 1.2],[1.2, 1.2, -.2, -.2],('1234'), ...
'HorizontalAlignment','center')
axis([-1 2 -1 2],'off')

% Draw a picture showing the adjacency matrix.
subplot(1,2,2);
xtemp = repmat(1:4,1,4);
ytemp = reshape(repmat(1:4,4,1),16,1)';
text(xtemp-.5,ytemp-.5,char('0'+A(:)),'HorizontalAlignment','center');
line([.25 0 0 .25 NaN 3.75 4 4 3.75],[0 0 4 4 NaN 0 0 4 4])
axis off tight
```

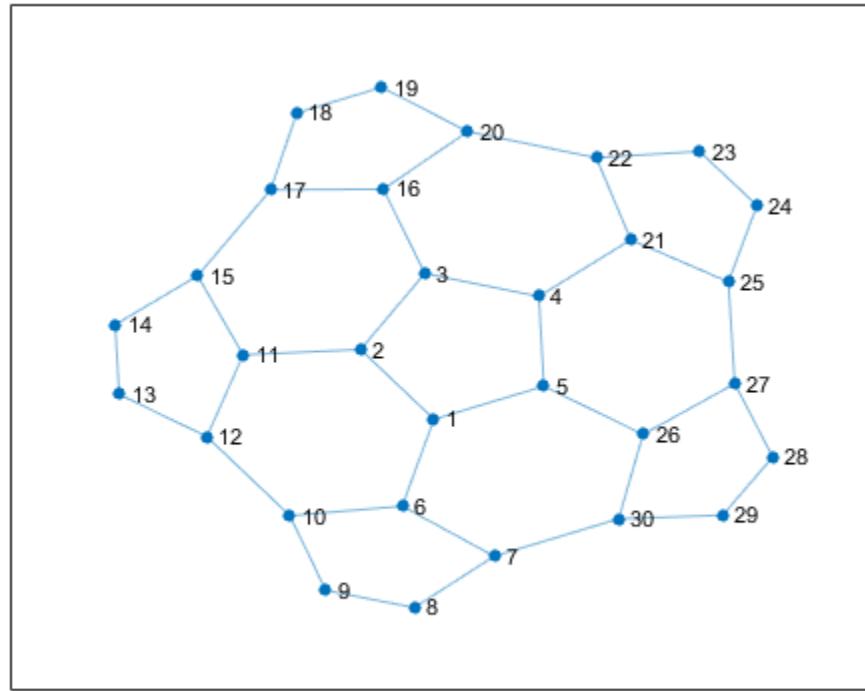


稀疏矩阵特别有助于表示非常大的图形。这是因为每个节点通常只会连接到少数几个其他节点。因此，对于大型图形，邻接矩阵中非零项的密度通常比较小。布基球邻接矩阵就是一个很好的示例，因为它是一个 60×60 的对称稀疏矩阵，仅包含 180 个非零元素。此矩阵的密度仅为 5%。

由于邻接矩阵定义图形，因此您可以使用邻接矩阵中的条目子集来绘制布基球的一部分。

使用 **adjacency** 函数为图形创建新的邻接矩阵。通过对该邻接矩阵进行索引创建一个新的较小图形，以显示布基球的一个半球中的节点。

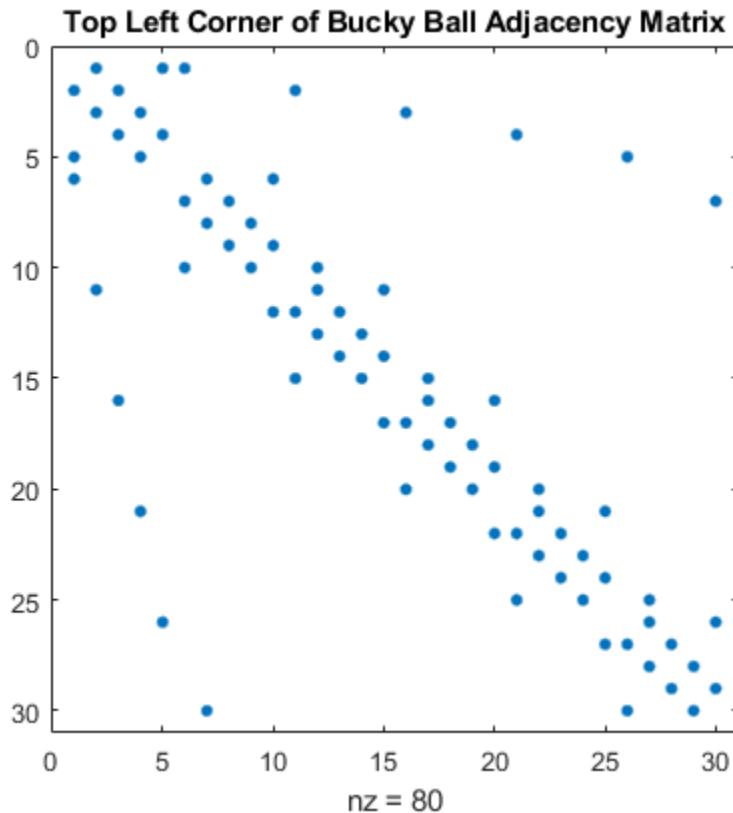
```
figure
A = adjacency(G);
H = graph(A(1:30,1:30));
h = plot(H);
```



要实现此半球的邻接矩阵可视化，请使用 `spy` 函数对邻接矩阵中的非零元素的轮廓进行绘图。

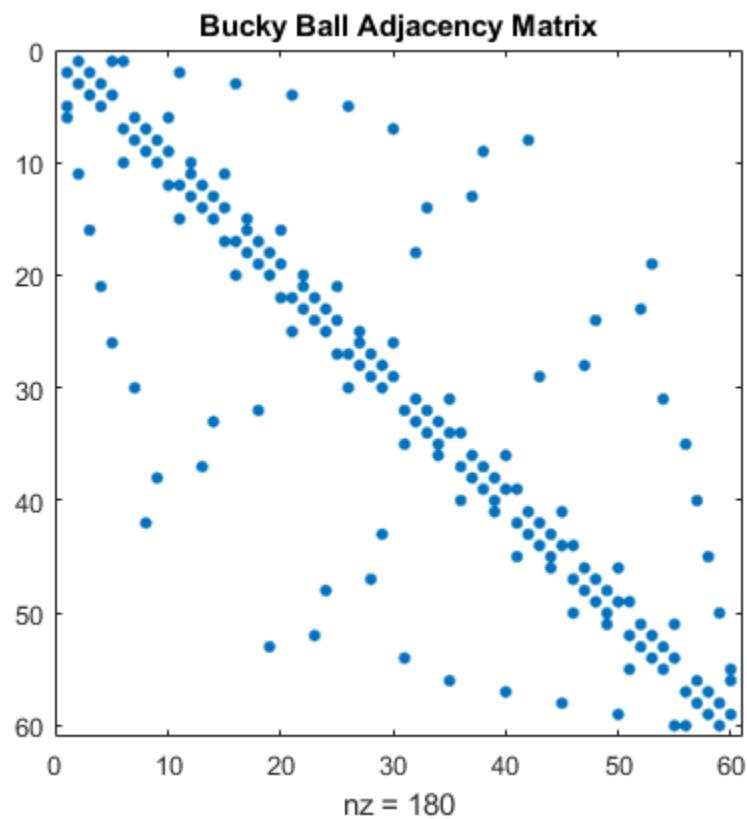
请注意，该矩阵是对称的，因此如果节点 i 与节点 j 相连，则节点 j 也与 i 相连。

```
spy(A(1:30,1:30))
title('Top Left Corner of Bucky Ball Adjacency Matrix')
```



最后，下面展示了整个布基球邻接矩阵的 spy 绘图。

```
spy(A)
title('Bucky Ball Adjacency Matrix')
```



另请参阅

`graph | spy`

稀疏矩阵重新排序

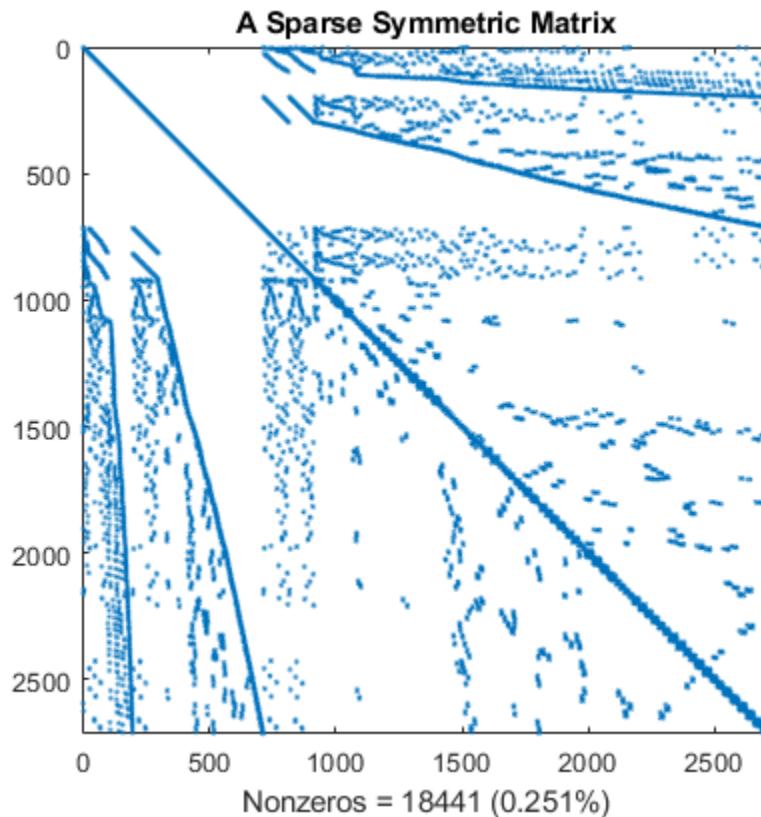
此示例说明对稀疏矩阵的各行和列重新排序可能会影响矩阵运算所需的速度和存储空间要求。

可视化稀疏矩阵

`spy` 图可以显示矩阵中的非零元素。

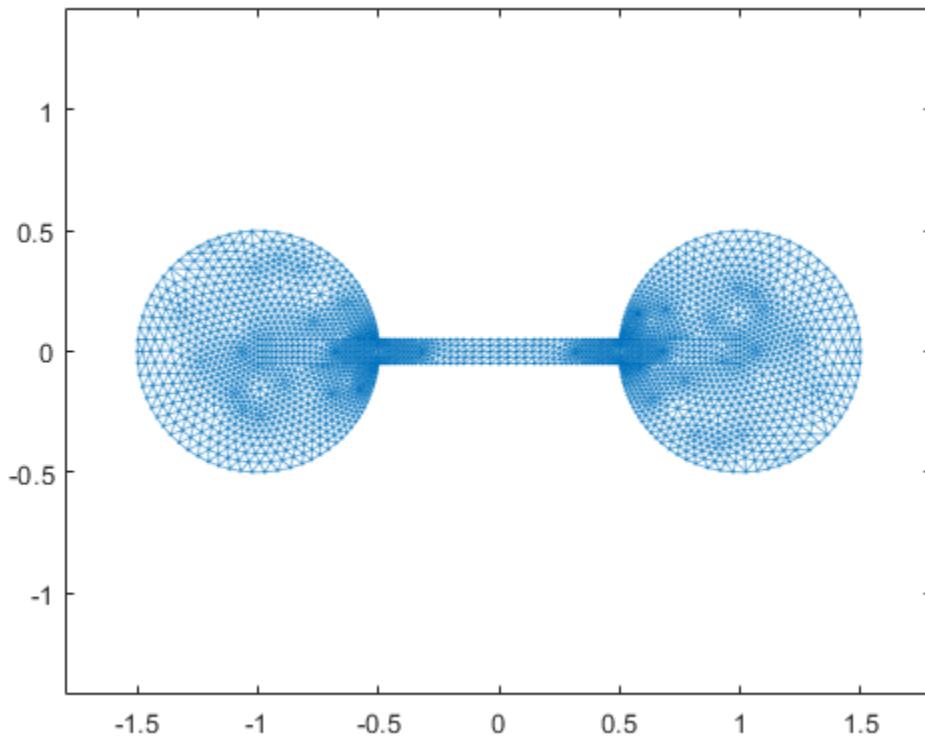
下面的 `spy` 图显示了从杠铃矩阵的一部分得到的稀疏对称正定矩阵。此矩阵描述类似杠铃的图中的连接。

```
load barbellgraph.mat
S = A + speye(size(A));
pct = 100 / numel(A);
spy(S)
title('A Sparse Symmetric Matrix')
nz = nnz(S);
xlabel(sprintf('Nonzeros = %d (%.3f%%)', nz, nz*pct));
```



以下是杠铃图。

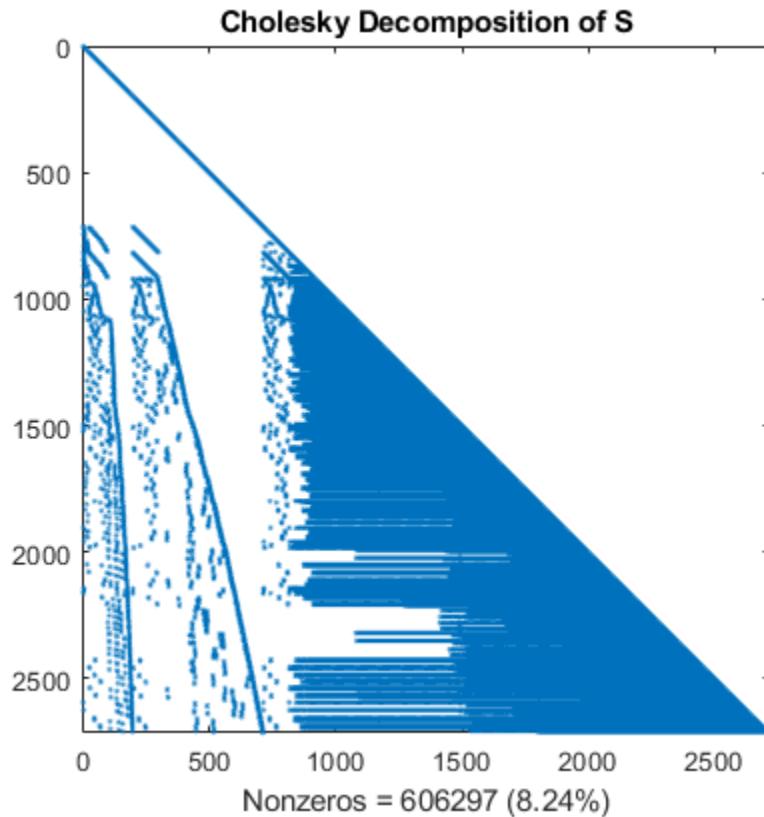
```
G = graph(S,'omitselfloops');
p = plot(G,'XData',xy(:,1),'YData',xy(:,2),'Marker','.');
axis equal
```



计算 Cholesky 因子

计算 Cholesky 因子 L , 其中 $S = L \cdot L'$ 。请注意, L 比未分解的 S 包含更多的非零元素, 因为计算 Cholesky 分解产生了填充非零值。这些填充值会降低算法速度并增加存储成本。

```
L = chol(S,'lower');
spy(L)
title('Cholesky Decomposition of S')
nc(1) = nnz(L);
xlabel(sprintf('Nonzeros = %d (%.2f%%)',nc(1),nc(1)*pct));
```



重新排序以加快计算速度

通过对矩阵的各行和各列重新排序，有可能减少由于分解而产生的填充值数量，从而降低后续计算的时间和存储成本。

MATLAB® 支持若干种不同的重新排序方法：

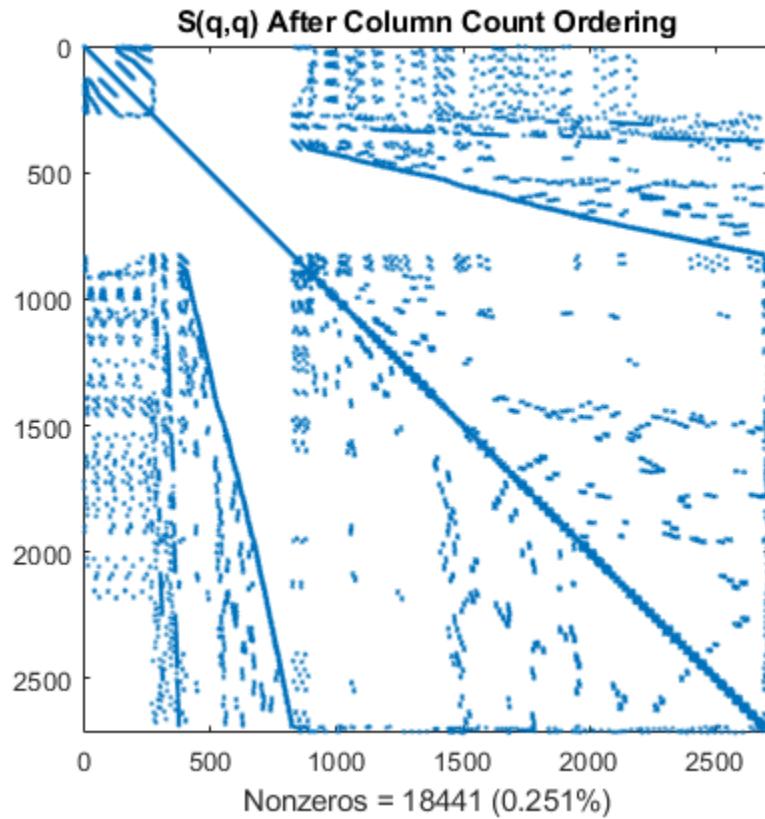
- **colperm**:列计数
- **symrcm**:反向 Cuthill-McKee 排序
- **amd**:最小度
- **dissect**:嵌套剖分

测试这些稀疏矩阵重新排序方法对杠铃矩阵的影响。

列计数重新排序

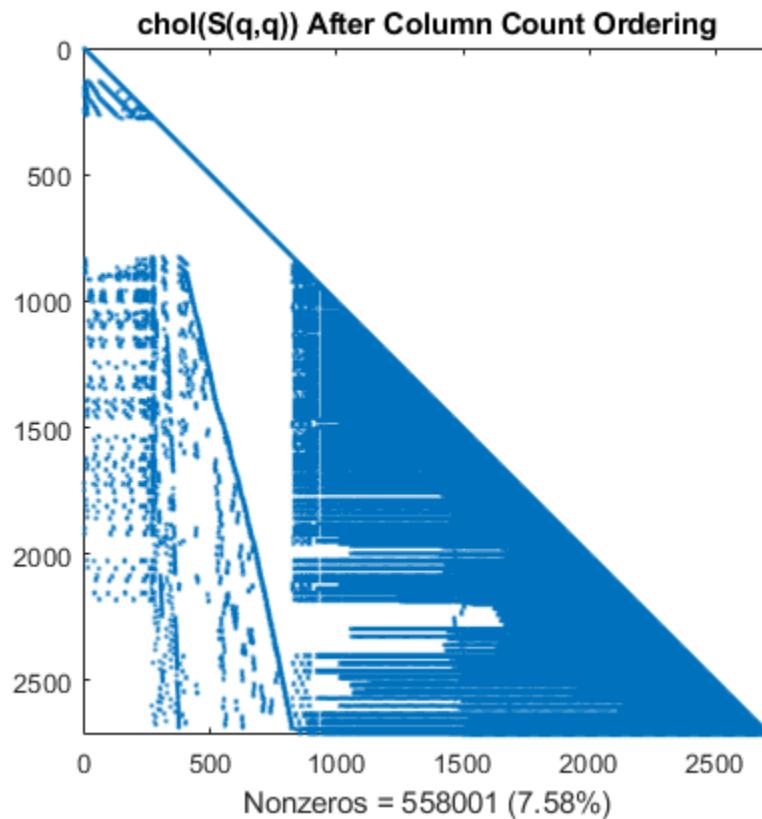
colperm 命令使用列计数重新排序算法将非零计数较大的行和列移向矩阵的末尾。

```
q = colperm(S);
spy(S(q,q))
title('S(q,q) After Column Count Ordering')
nz = nnz(S);
xlabel(sprintf('Nonzeros = %d (%.3f%%)',nz,nz*pct));
```



对于此矩阵，列计数排序几乎无法减少 Cholesky 分解的时间和存储量。

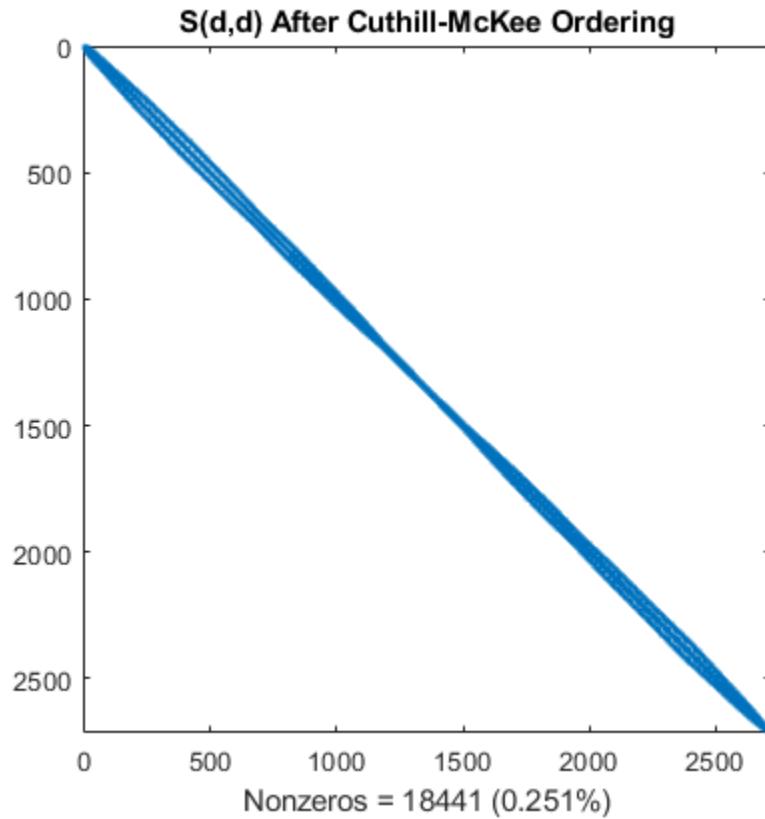
```
L = chol(S(q,q),'lower');
spy(L)
title('chol(S(q,q)) After Column Count Ordering')
nc(2) = nnz(L);
xlabel(sprintf('Nonzeros = %d (%.2f%%)',nc(2),nc(2)*pct));
```



反向 Cuthill-McKee 重新排序

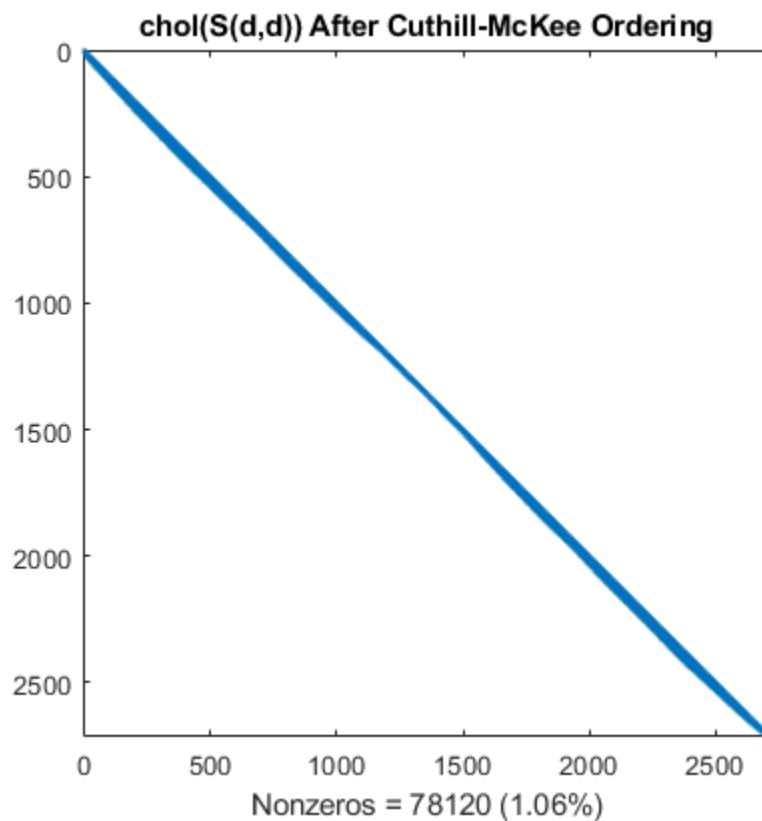
`symrcm` 命令使用反向 Cuthill-McKee 重新排序算法将所有非零元素移至更靠近对角线的位置，从而减小原始矩阵的带宽。

```
d = symrcm(S);
spy(S(d,d))
title('S(d,d) After Cuthill-McKee Ordering')
nz = nnz(S);
xlabel(sprintf('Nonzeros = %d (%.3f%%)',nz,nz*pct));
```



Cholesky 分解产生的填充值被限制在该带宽内，因此分解重新排序后的矩阵需要的时间和存储空间较少。

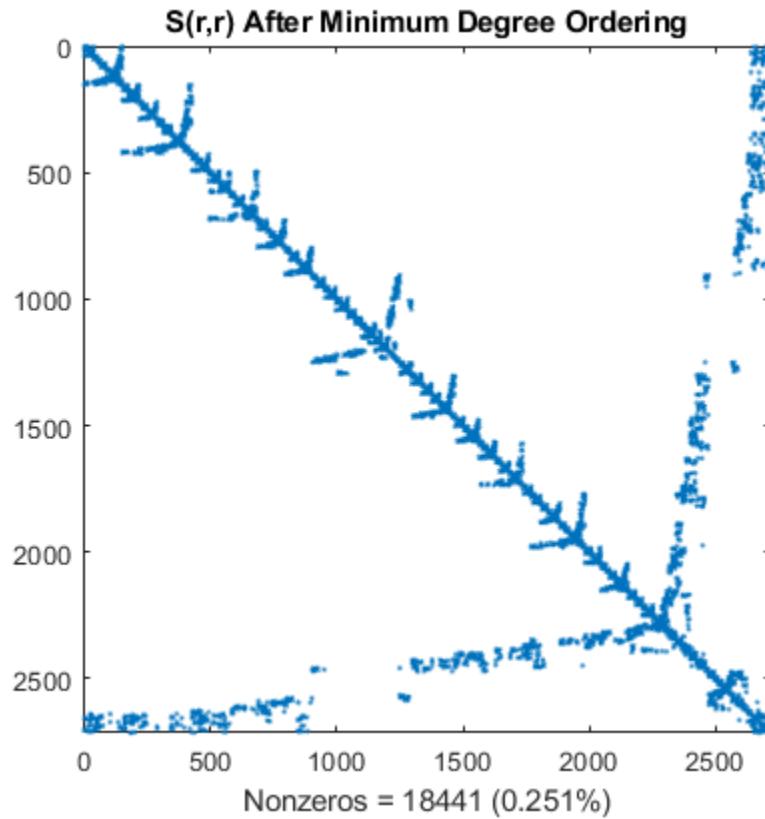
```
L = chol(S(d,d),'lower');
spy(L)
title('chol(S(d,d)) After Cuthill-McKee Ordering')
nc(3) = nnz(L);
xlabel(sprintf('Nonzeros = %d (%.2f%%)', nc(3),nc(3)*pct));
```



最小度重新排序

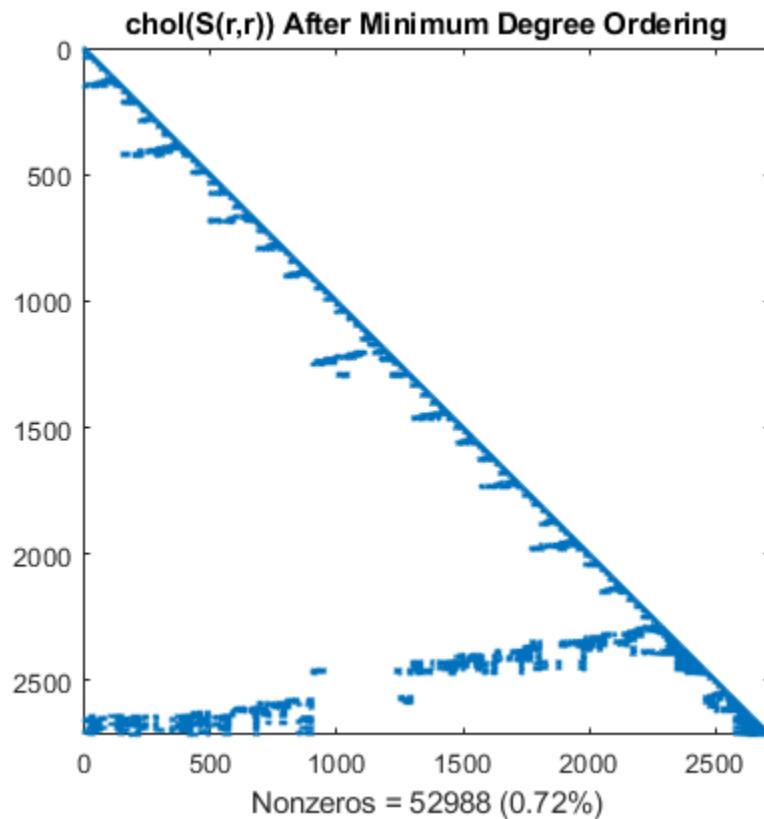
amd 命令使用一种近似最小度算法（一种非常有用的图论技巧）在矩阵中产生大块的零。

```
r = amd(S);
spy(S(r,r))
title('S(r,r) After Minimum Degree Ordering')
nz = nnz(S);
xlabel(sprintf('Nonzeros = %d (%.3f%%)',nz,nz*pct));
```



Cholesky 分解会保留最小度算法产生的各块零。此结构可以显著降低时间和存储成本。

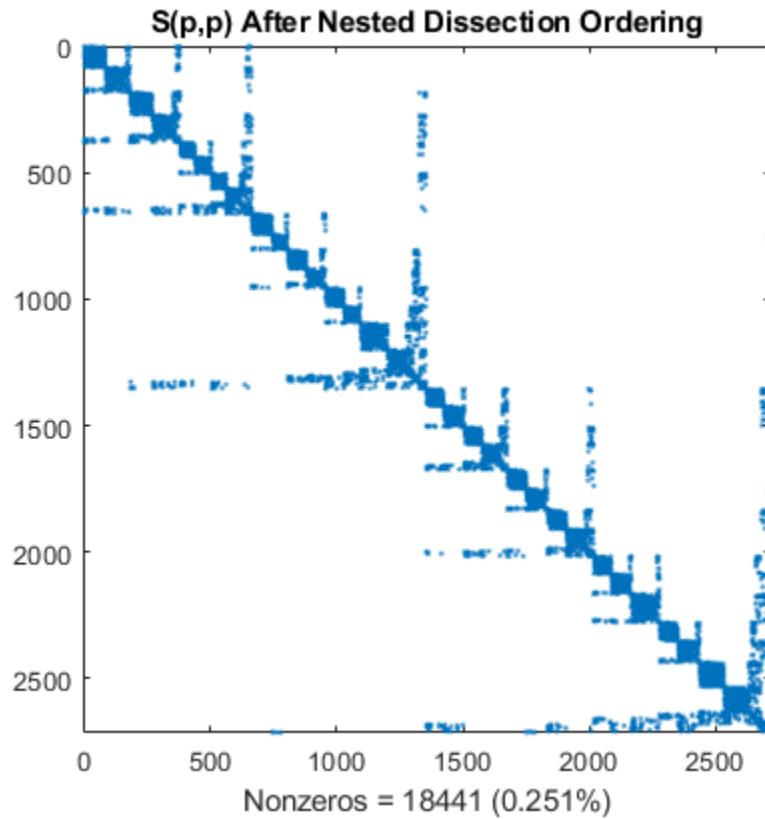
```
L = chol(S(r,r),'lower');
spy(L)
title('chol(S(r,r)) After Minimum Degree Ordering')
nc(4) = nnz(L);
xlabel(sprintf('Nonzeros = %d (%.2f%%)',nc(4),nc(4)*pct));
```



嵌套剖分置换

`dissect` 函数使用图论方法来生成减少填充的排序。该算法将矩阵视为图的邻接矩阵，通过折叠顶点和边来粗化图，重排较小的图，然后通过细化步骤对小图去粗，得到重排的原始图。结果将得到一个功能强大的算法，与其他重新排序算法相比，它往往产生最少的填充量。

```
p = dissect(S);
spy(S(p,p))
title('S(p,p) After Nested Dissection Ordering')
nz = nnz(S);
xlabel(sprintf('Nonzeros = %d (%.3f%%)',nz,nz*pct));
```

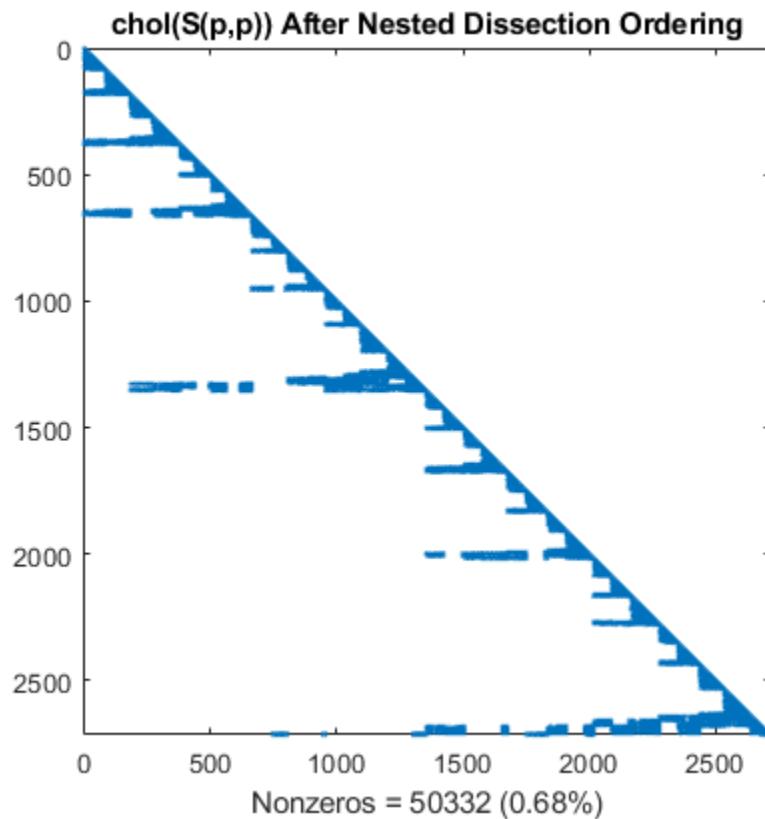


与最小度排序相似，嵌套剖分排序的 Cholesky 分解最大程度地保留主对角线下方的 $S(d,d)$ 非零结构体。

```

L = chol(S(p,p),'lower');
spy(L)
title('chol(S(p,p)) After Nested Dissection Ordering')
nc(5) = nnz(L);
xlabel(sprintf('Nonzeros = %d (%.2f%%)',nc(5),nc(5)*pct));

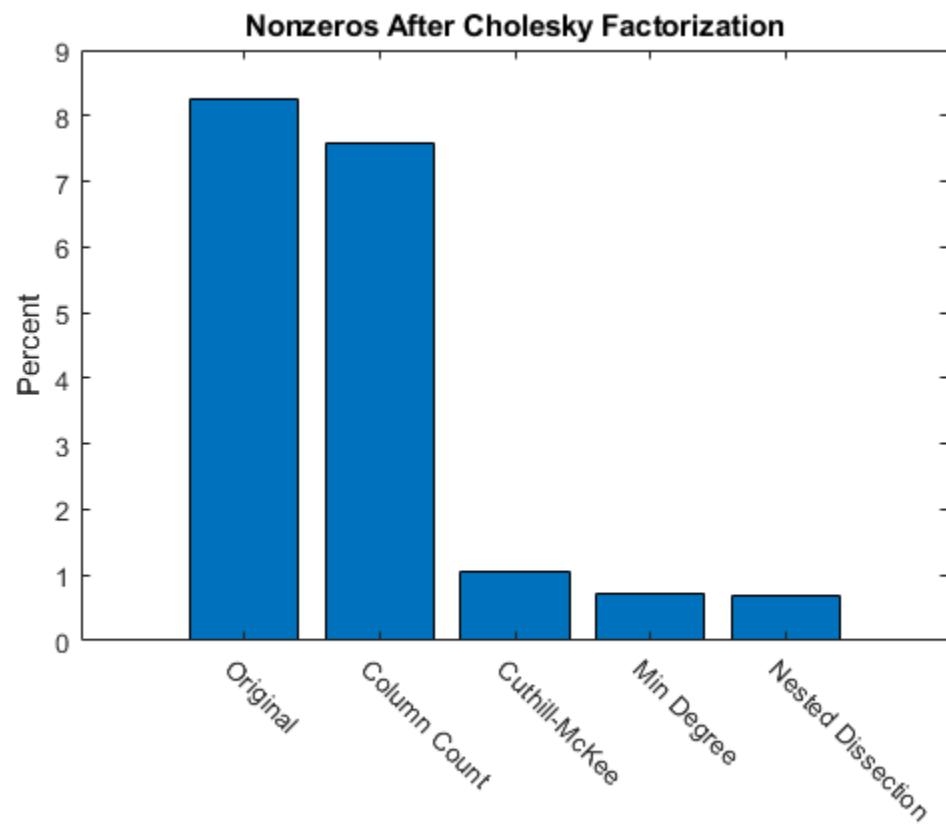
```



汇总结果

下面的条形图总结了执行 Cholesky 分解之前对矩阵重新排序所带来的影响。虽然原始矩阵的 Cholesky 分解约有 8% 的元素非零，但使用 `dissect` 或 `symamd` 可将该密度降至低于 1%。

```
labels = {'Original','Column Count','Cuthill-McKee',...
    'Min Degree','Nested Dissection'};
bar(nc*pct)
title('Nonzeros After Cholesky Factorization')
ylabel('Percent');
ax = gca;
ax.XTickLabel = labels;
ax.XTickLabelRotation = -45;
```



另请参阅

[chol](#) | [colperm](#) | [nnz](#) | [spy](#) | [symamd](#) | [symrem](#)

线性方程组的迭代方法

数值线性代数最重要也是最常见的应用之一是可求解以 $A \cdot x = b$ 形式表示的线性方程组。当 A 为大型稀疏矩阵时，您可以使用迭代方法求解线性方程组，使用这一方法，您可在计算的运行时间与解的精度之间进行权衡。本主题介绍 MATLAB 中可用于求解方程 $A \cdot x = b$ 的迭代方法。

直接方法与迭代方法

求解线性方程 $A \cdot x = b$ 的方法有两种：

- **直接方法**是高斯消去法的变体。这些方法通过矩阵运算（例如 LU、QR 或 Cholesky 分解）直接涉及各个矩阵元素。您可以使用直接方法来求解具有高精度水平的线性方程，但在大型稀疏矩阵上运行这些方法时，速度可能会很慢。用直接方法求解线性方程组的速度很大程度上取决于系数矩阵的大小。

例如，以下代码用于求解一个较小的线性方程组。

```
A = magic(5);
b = sum(A,2);
x = A\b;
norm(A*x-b)
```

ans =

1.4211e-14

MATLAB 通过矩阵除法运算符 / 和 \ 以及 lsqminnorm、decomposition 和 linsolve 等函数实现直接的方法。

- **迭代方法**在经过有限的步数之后生成线性方程组的逼近解。这些方法对大型方程组非常有用，在求解这些方程组时，通过牺牲精度来缩短运行时间是合理的。这些方法仅间接涉及系数矩阵、整个矩阵-向量积或抽象的线性运算函数。迭代方法通常仅适用于稀疏矩阵，因为较小的方程组可以使用直接方法轻松求解。使用间接方法求解线性方程组的速度不像直接方法那样严重依赖于系数矩阵的大小。但是，使用迭代方法通常需要针对每个特定问题调优参数。

例如，以下代码用于求解一个具有对称正定系数矩阵的大型稀疏线性方程组。

```
A = delsq(numgrid('L',400));
b = ones(size(A,1),1);
x = pcg(A,b,[],1000);
norm(A*x-b)
```

pcg converged at iteration 796 to a solution with relative residual 9.9e-07.

ans =

3.4285e-04

MATLAB 根据系数矩阵 A 的属性，实现了多种具有不同优缺点的迭代方法。

如果提供实施这些方法所需的足够空间，直接方法与间接方法相比，通常前者更快，适用性更广。通常，您应该先尝试使用 $x = A \cdot b$ 。如果直接求解的速度太慢，则可以尝试使用迭代方法。

常规迭代算法

求解线性方程组的大多数迭代算法都遵循类似的过程：

- 1 首先获取解向量 x_0 的初始估计值。 (除非您指定了更好的估计值, 否则通常为零元素向量。)
- 2 计算残差范数 $res = \text{norm}(b - A*x_0)$ 。
- 3 将残差与指定的容差进行比较。如果 $res \leq tol$, 则结束计算并返回 x_0 的计算答案。
- 4 根据残差值和其他计算出的量来更新向量 x_0 的幅值和方向。
- 5 重复步骤 2 到 4, 直到 x_0 的值足以满足容差要求。

迭代方法在步骤 4 中更新 x_0 的幅值和方向的方式各有不同, 而且某些迭代方法在步骤 2 和 3 中的收敛条件也略有差别, 以下概括了所有迭代求解器遵循的基本过程。

迭代方法概要

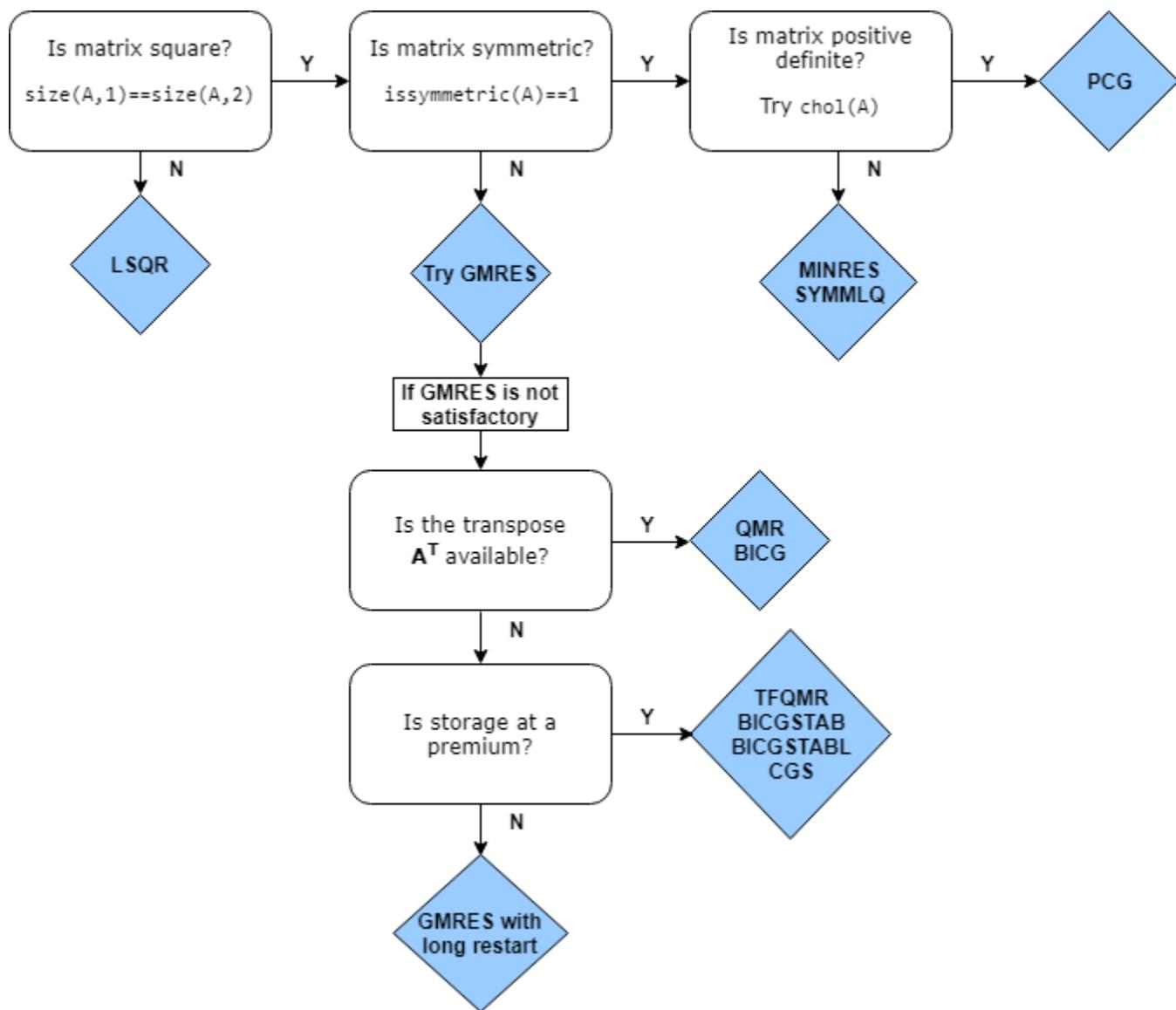
MATLAB 有若干函数用于实现稀疏线性方程组的迭代方法。这些方法设计用于对 $Ax = b$ 求解或求 $\|b - Ax\|$ 范数的最小值。其中几种方法非常相似, 并且基于相同的基础算法, 但每种算法在特定情况下有各自的好处 [1], [2]。

说明	注释
lsqr (最小二乘法)	<ul style="list-style-type: none"> • 系数矩阵可以是矩形。 • 唯一适用于矩形方程组的求解器。 • 在解析上等效于应用到标准方程 $(A'*A)*x = A'*b$ 的共轭梯度法 (PCG), 但具有更好的数值属性。
pcg (预条件共轭梯度法)	<ul style="list-style-type: none"> • 系数矩阵必须是对称正定矩阵。 • 由于只需要存储有限数量的向量, 因此是最有效的对称正定方程组求解器。
minres (最小残差法)	<ul style="list-style-type: none"> • 系数矩阵必须是对称矩阵, 但无需是正定矩阵。 • 每次迭代都会最小化 2-范数中的残差, 从而保证算法能够逐步取得进展。 • 不会出现算法崩溃 (算法无法逼近解而停止执行)。
symmlq (对称 LQ)	<ul style="list-style-type: none"> • 系数矩阵必须是对称矩阵, 但无需是正定矩阵。 • 求解投影方程组, 并使残差与所有先前的残差正交。 • 不会出现算法崩溃 (算法无法逼近解而停止执行)。
bicg (双共轭梯度法)	<ul style="list-style-type: none"> • 系数矩阵必须是正方形的, 但不必对称。 • bicg 的计算成本低, 但收敛不规则而且不可靠。 • 虽然对大多数问题而言, bicg 很少是最佳选择, 但它具有很重要的历史地位, 因为很多算法都是基于它改进而来。

说明	注释
bicgstab (双共轭梯度稳定法)	<ul style="list-style-type: none"> 系数矩阵必须是正方形的，但不必对称。 交替使用 BiCG 步骤与 GMRES(1) 步骤，以提高稳定性。
bicgstabl (双共轭梯度稳定法 (l))	<ul style="list-style-type: none"> 系数矩阵必须是正方形的，但不必对称。 交替使用 BiCG 步骤与 GMRES(2) 步骤，以提高稳定性。
cgs (共轭梯度二乘法)	<ul style="list-style-type: none"> 系数矩阵必须是正方形的，但不必对称。 需要与 bicg 相同的每次迭代操作数，但要避免通过操作平方残差来使用转置。
gmres (广义最小残差法)	<ul style="list-style-type: none"> 系数矩阵必须是正方形的，但不必对称。 最可靠的算法之一，因为每次迭代都能最小化残差范数。 所需的工作和存储量随迭代次数线性增加。 选择适当的 restart 值至关重要，以避免不必要的工作和存储。
qmr (拟最小残差法)	<ul style="list-style-type: none"> 系数矩阵必须是正方形的，但不必对称。 每次迭代的开销略高于 bicg，但提供了更好的稳定性。
tfqmr (无转置拟最小残差法)	<ul style="list-style-type: none"> 系数矩阵必须是正方形的，但不必对称。 当内存有限时，是尝试用于求解对称不定方程组的最佳求解器。

选择迭代求解器

以下是 MATLAB 中的迭代求解器的流程图，大致概括了每种求解器适用的情况。一般而言，您可以对几乎所有的二次方非对称问题使用 **gmres**。在一些情况下，双共轭梯度算法 (**bicg**、**bicgstab**、**cgs** 等) 的效率高于 **gmres**，但它们的收敛行为难以预测，因此 **gmres** 往往是更好的初始选择。



预条件子

迭代方法的收敛速度取决于系数矩阵的频谱（特征值）。因此，您可以通过将系数矩阵变换为具有更好的频谱（聚类特征值或接近 1 的条件数）来改善大多数迭代方法的收敛性和稳定性。这种变换的执行方法是将第二个被称为预条件子的矩阵应用于方程组。此过程将线性方程组

$$Ax = b$$

变换为等效方程组

$$\tilde{A}\tilde{x} = \tilde{b} .$$

理想的预条件子将系数矩阵 A 变换为单位矩阵，因为任何迭代方法都将在使用这种预条件子的一次迭代中实现收敛。实际上，寻找一个好的预条件子需要权衡。采用以下三种方式之一执行变换：左侧预条件、右侧预条件或拆分预条件。

第一种情况称为左侧预条件，因为预条件子矩阵 M 出现在 A 的左侧：

$$(M^{-1}A)x = (M^{-1}b).$$

以下迭代求解器使用左侧预条件：

- **gmres**
- **qmr**
- **bicg**

在右侧预条件中， M 出现在 A 的右侧：

$$(AM^{-1})(Mx) = b.$$

以下迭代求解器使用右侧预条件：

- **lsqr**
- **tfqmr**
- **bicgstab**
- **bicgstabl**
- **cgs**

最后，对于对称系数矩阵，拆分预条件可确保新方程组仍具有对称矩阵。预条件子 $M = HH^T$ 同时出现在 A 的两侧：

$$(H^{-1}AH^{-T})H^Tx = (H^{-1}b)$$

经过拆分预条件后的方程组的求解器算法以上述方程为基础，但实际上无需计算 H 。求解器算法仅仅直接应用 M 或 $\text{inv}(M)$ 。

以下迭代求解器使用拆分预条件：

- **pcg**
- **minres**
- **symmlq**

在所有情况下，选用预条件子 M 以加快迭代方法的收敛。当迭代解的残差停滞不前或两次迭代之间进展甚微时，通常意味着您需要生成一个预条件子矩阵以加入问题中。

MATLAB 中的迭代求解器允许您指定单个预条件子矩阵 M ，或两个预条件子矩阵因子，使得 $M = M_1M_2$ 。这样就能轻松指定通过分解计算出的预条件子，例如 $M = LU$ 。请注意，在同时保留了 $M = HH^T$ 的拆分预条件情况下， $M1$ 和 $M2$ 输入与 H 因子之间没有直接关联。

在许多情况下，预条件子在给定问题的数学模型中自然发生。例如，带有可变系数的偏微分方程可以近似地用带有常系数的偏微分方程表示。在没有自然预条件子的情况下，可以使用下表中的不完全分解之一。

函数	分解	说明
ilu	$A \approx LU$	正方形或矩形矩阵的不完全 LU 分解。

函数	分解	说明
ichol	$A \approx LL^*$	对称正定矩阵的不完全 Cholesky 分解。

有关 ilu 和 ichol 的详细信息, 请参阅 “不完全分解” (第 4-20 页)。

预条件子示例

以二维方形域中的拉普拉斯方程的五点有限差分逼近方程为例。以下命令使用预条件共轭梯度法 (PCG) 预条件子 $M = L^*L'$, 其中 L 是 A 的零填充不完全 Cholesky 因子。

```
A = delsq(numgrid('S',250));
b = ones(size(A,1),1);
tol = 1e-3;
maxit = 100;
L = ichol(A);
x = pcg(A,b,tol,maxit,L,L');

pcg converged at iteration 92 to a solution with relative residual 0.00076.
```

pcg 需要 92 次迭代才能达到指定的容差。但是, 使用其他预条件子可以产生更好的结果。例如, 使用 ichol 构造修正的不完全 Cholesky, pcg 只需经过 39 次迭代便可达到指定的容差。

```
L = ichol(A,struct('type','nofill','michol','on'));
x = pcg(A,b,tol,maxit,L,L');

pcg converged at iteration 39 to a solution with relative residual 0.00098.
```

均衡和重新排序

对于难以计算的问题, 您可能需要比 ilu 或 ichol 直接生成的预条件子更好的预条件子。例如, 您可能希望生成质量更好的预条件子, 或最小化要执行的计算量。在这些情况下, 可以使用均衡来使系数矩阵更加对角占优 (这可以生成质量更好的预条件子), 并使用重新排序来最小化矩阵因子中的非零元素数量 (这可以降低内存需求并可能提高后续计算的效率)。

如果同时使用均衡和重新排序来生成预条件子, 则该过程为:

- 1 对系数矩阵使用 equilibrate。
- 2 使用稀疏矩阵重新排序函数 (例如 dissect 或 symrcm) 对均衡后的矩阵重新排序。
- 3 使用 ilu 或 ichol 生成最终预条件子。

以下示例使用均衡和重新排序生成了用于稀疏系数矩阵的预条件子。

- 1 创建系数矩阵 A 和由 1 组成的向量 b 作为线性方程的右侧。计算 A 的条件数估计值。

```
load west0479
A = west0479;
n = size(A,1);
b = ones(n,1);
condest(A)

ans =
```

1.4244e+12

使用 equilibrate 来改善系数矩阵的条件数。

```
[P,R,C] = equilibrate(A);
Anew = R*P*A*C;
bnew = R*P*b;
condest(Anew)
```

ans =

5.1042e+04

- 2 使用 `dissect` 对均衡后的矩阵重新排序。

```
q = dissect(Anew);
Anew = Anew(q,q);
bnew = bnew(q);
```

- 3 使用不完全 LU 分解生成预条件子。

`[L,U] = ilu(Anew);`

- 4 使用 `gmres`, 通过预条件子矩阵、容差 `1e-10`、最大外部迭代次数 50 和内部迭代次数 30, 求解线性方程组。

```
tol = 1e-10;
maxit = 50;
restart = 30;
[xnew, flag, relres] = gmres(Anew,bnew,restart,tol,maxit,L,U);
x(q) = xnew;
x = C*x(:);
```

现在, 将 `gmres` (其中包括预条件子) 返回的 `relres` 相对残差与没有使用预条件子的相对残差 `resnew` 和没有经过均衡的相对残差 `res` 进行比较。结果表明, 尽管线性方程组都是等效的, 但不同的方法会将不同的权重应用于每个元素, 而这可能显著影响残差值。

```
relres
resnew = norm(Anew*xnew - bnew) / norm(bnew)
res = norm(A*x - b) / norm(b)

relres =
8.7537e-11
resnew =
3.6805e-08
res =
5.1415e-04
```

使用线性运算函数取代矩阵

MATLAB 中的迭代求解器不要求为 `A` 提供数值矩阵。由于求解器执行的计算主要使用矩阵-向量乘法 `A*x` 或 `A'*x` 的结果, 因此您可以改而提供一个函数来计算这些线性运算的结果。计算这些量的函数通常被称为线性运算函数。

除了使用线性运算函数取代系数矩阵 `A`, 还可以使用线性运算函数取代预条件子矩阵 `M`。在这种情况下, 函数需要计算 `M\x` 或 `M'\x`。

通过使用线性运算函数, 您可以利用 `A` 或 `M` 中的模式来计算线性运算的值, 这比求解器执行满矩阵-向量乘法的效率更高。这也意味着, 您不需要内存来存储系数矩阵或预条件子矩阵, 因为线性运算函数通常计算矩阵-向量乘法的结果而根本不用构建矩阵。

例如, 有以下系数矩阵

```
A = [2 -1 0 0 0 0;
      -1 2 -1 0 0 0;
```

```
0 -1 2 -1 0 0;  
0 0 -1 2 -1 0;  
0 0 0 -1 2 -1;  
0 0 0 0 -1 2];
```

当此系数矩阵乘以一个向量时，仅有的非零元素是那些与 A 的三对角元素相乘的元素。因此，对于给定的向量 \mathbf{x} ，线性运算函数只需要将三个向量相加便可计算 $\mathbf{A}^*\mathbf{x}$ 的值：

```
function y = linearOperatorA(x)  
y = -1*[0; x(1:end-1)] ...  
+ 2*x ...  
+ -1*[x(2:end); 0];  
end
```

大多数迭代求解器要求线性运算函数基于 A 返回 $\mathbf{A}^*\mathbf{x}$ 的值。同样，对于预条件子矩阵 M，该函数通常必须计算 $M \setminus \mathbf{x}$ 。对于求解器 lsqr、qmr 和 bicg，如果请求了返回 $\mathbf{A}'^*\mathbf{x}$ 或 $M'^\setminus \mathbf{x}$ 的值，线性运算函数还需返回这些值。有关线性运算函数的示例和说明，请参阅迭代求解器参考页。

参考

- [1] Barrett, R., M. Berry, T. F. Chan, et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1994.
- [2] Saad, Youcef, *Iterative Methods for Sparse Linear Equations*. PWS Publishing Company, 1996.

另请参阅

详细信息

- “线性方程组”（第 2-10 页）

图和网络算法

- “有向图和无向图”（第 5-2 页）
- “修改现有图的节点和边”（第 5-10 页）
- “添加图节点名称、边权重和其他属性”（第 5-13 页）
- “图的绘制和自定义”（第 5-17 页）
- “可视化广度优先搜索和深度优先搜索”（第 5-28 页）
- “使用拉普拉斯矩阵为图分区”（第 5-33 页）
- “将节点属性添加到图论图数据游标”（第 5-36 页）
- “生成 Watts-Strogatz 小世界图形模型”（第 5-39 页）
- “使用 PageRank 算法对网站进行排名”（第 5-46 页）
- “为图节点和边添加标签”（第 5-53 页）

有向图和无向图

本节内容

- “什么是图？”（第 5-2 页）
- “创建图”（第 5-5 页）
- “图节点 ID”（第 5-8 页）
- “修改或查询现有图”（第 5-8 页）

什么是图？

图是表示各种关系的节点和边的集合：

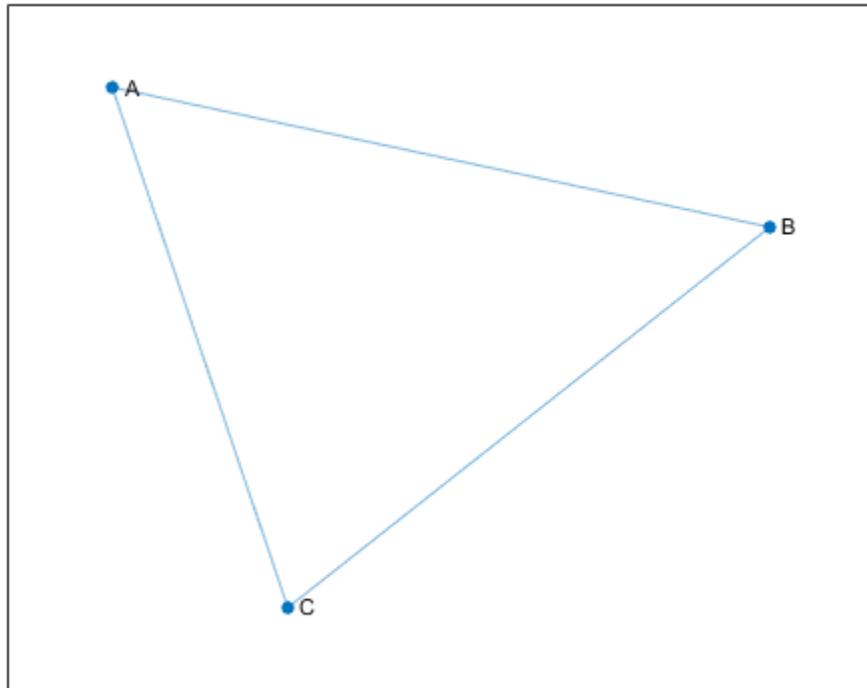
- **节点**是与对象对应的顶点。
- **边**是对象之间的连接。
- 图的边有时会有**权重**，表示节点之间的每个连接的强度（或一些其他属性）。

这些定义是概括性的，因为节点和边在图中的确切含义取决于具体的应用情形。例如，您可以使用图为社交网站中的朋友关系建模。图节点表示人，边表示朋友关系。图与物理对象和各种情况的自然对应关系意味着，您可以使用图对各种系统进行建模。例如：

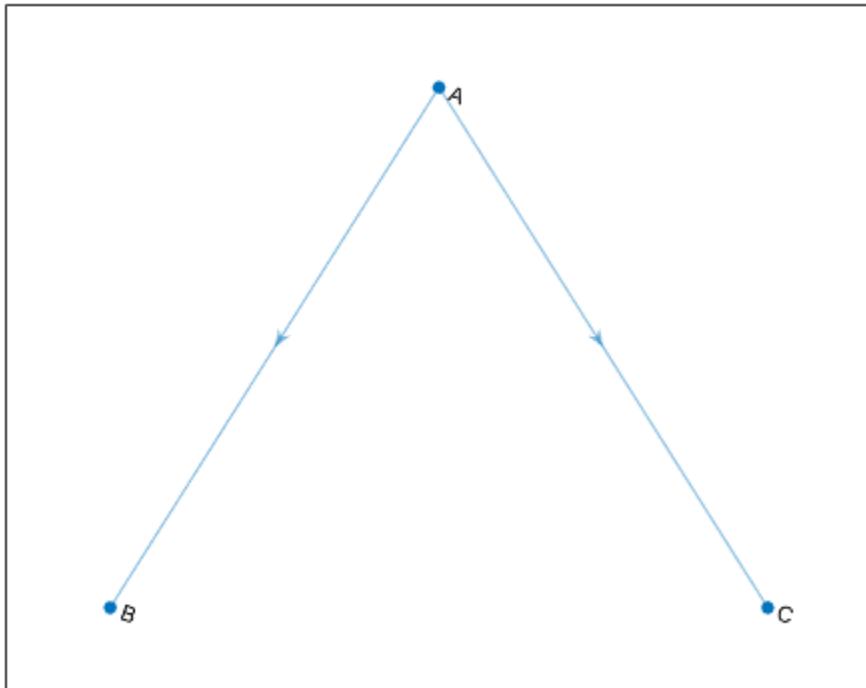
- 网页链接 - 图节点代表网页，边表示网页之间的超链接。
- 机场 - 图节点代表机场，边表示机场之间的航班。

在 MATLAB 中，**graph** 和 **digraph** 函数用于构建表示无向图和有向图的对象。

- **无向图**的边没有方向。这些边指示双向关系，因为每条边都可以在两个方向上穿过。下图显示了一个包含三个节点和三条边的简单无向图。



- **有向图**的边带有方向。这些边指示单向关系，因为每条边只能在单个方向上穿过。下图显示了一个包含三个节点和两条边的简单有向图。



边在图中的确切位置、长度或方向通常没有含义。换言之，只要基础结构不变，就可以通过重新排列节点和/或使边扭曲，以多种不同的方式显示同一个图。

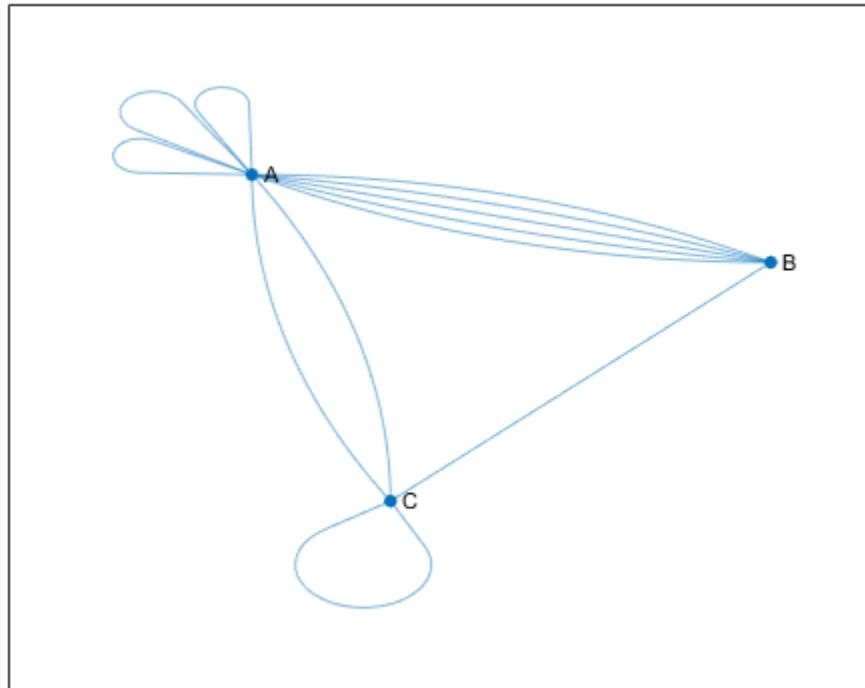
自环和多重图

使用 **graph** 和 **digraph** 创建的图可以有一个或多个自环，自环是指一条边的两端为同一个节点。此外，图可以具有多条有相同源节点和目标节点的边，这样的图称为多重图。多重图可能包含自环，也可能不包含。

对于 MATLAB 中的图算法函数来说，如果图中包含的节点只有一个自环，则不属于多重图。但是，如果图中包含的节点具有多个自环，则属于多重图。

例如，下图显示了具有多个自环的无向多重图。节点 A 有三个自环，节点 C 有一个。该图包含以下三个条件，任何一个条件都满足多重图的条件。

- 节点 A 有三个自环。
- 节点 A 和 B 之间有五条边。
- 节点 A 和 C 之间有两条边。



要确定给定的图是否为多重图，请使用 `ismultigraph` 函数。

创建图

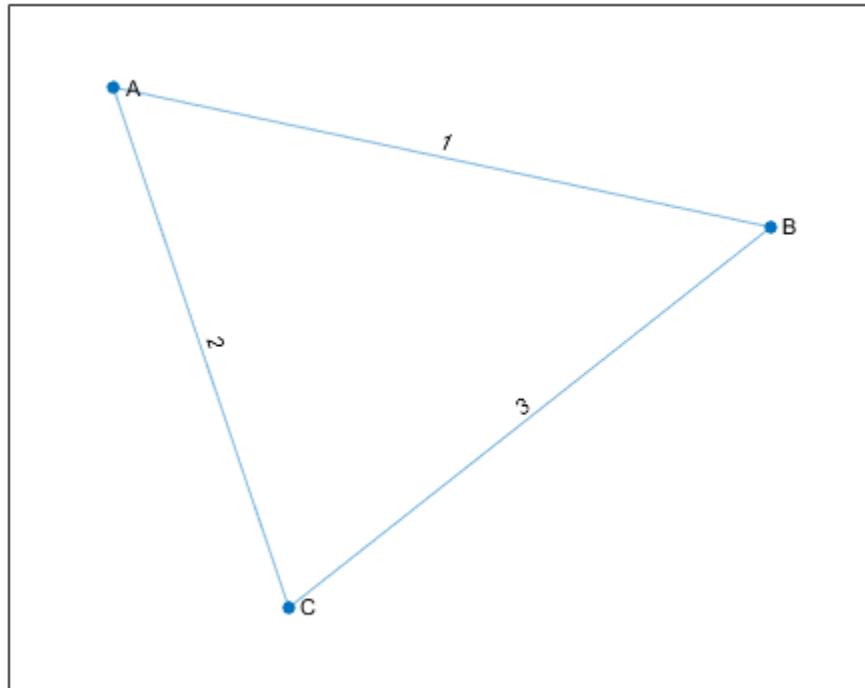
创建图的主要方式包括使用邻接矩阵或边列表。

邻接矩阵

有一种表示图中信息的方法是使用方形邻接矩阵。邻接矩阵中的非零项表示两个节点之间的边，条目值表示边的权重。邻接矩阵的对角线元素通常为零，但非零对角线元素表示自环或通过边与其自身相连的节点。

- 当您使用 `graph` 创建无向图时，邻接矩阵必须对称。但在实践中，为避免重复，这些矩阵通常为三角形。要仅使用邻接矩阵的上三角或下三角构建无向图，请使用 `graph(A,'upper')` 或 `graph(A,'lower')`。
- 当您使用 `digraph` 创建有向图时，邻接矩阵不需要对称。
- 对于大型图，邻接矩阵包含许多零并且通常为稀疏矩阵。
- 您不能从邻接矩阵创建多重图。

例如，考虑创建如下无向图。



可以通过下面的邻接矩阵表示该图：

$$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 3 & 0 \end{pmatrix} .$$

要在 MATLAB 中构建该图，请输入：

```
A = [0 1 2; 1 0 3; 2 3 0];
node_names = {'A','B','C'};
G = graph(A,node_names)
```

```
G =
```

```
graph with properties:
```

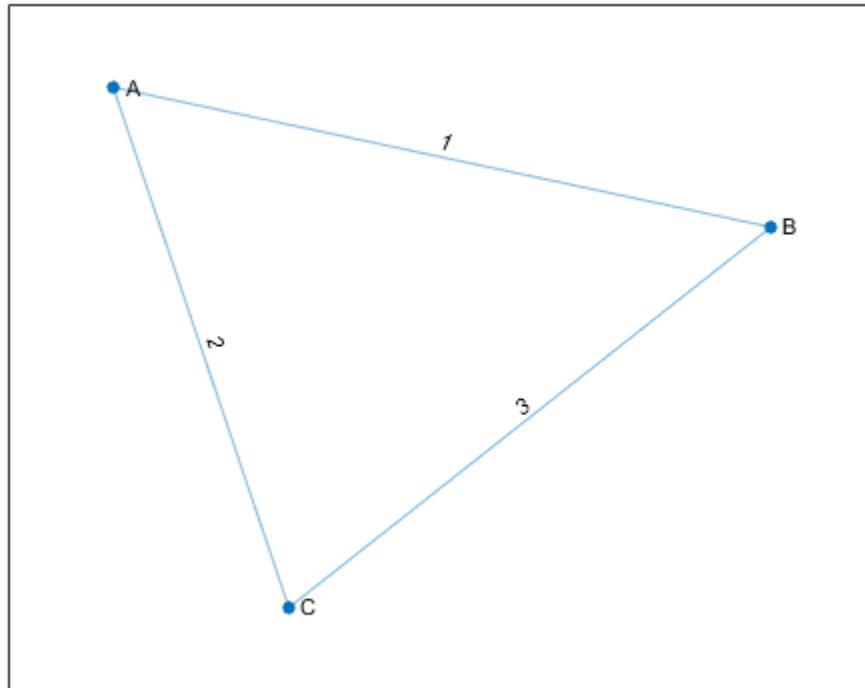
```
Edges: [3×2 table]
Nodes: [3×1 table]
```

您可以使用邻接矩阵通过 `graph` 或 `digraph` 函数来创建图，也可以使用 `adjacency` 函数求预先存在的图的加权或未加权的稀疏邻接矩阵。

边列表

表示图信息的另一种方法是列出所有边。

例如，考虑创建与上面相同的无向图。



现在用边列表表示该图

Edge	Weight
(A, B)	1
(A, C)	2
(B, C)	3

从边列表中很容易得出以下结论：该图包含三个唯一节点 A、B 和 C，这三个节点通过三条列出的边相连。如果该图有断开的节点，边列表中将不会列出这些节点，您需要单独指定它们。

在 MATLAB 中，边列表按列划分为源节点和目标节点。对于有向图，边的方向（从源到目标）很重要；但对于无向图，源节点和目标节点是可以互换的。使用边列表构建该图的一种方法是，对源节点、目标节点和边权重使用单独的输入：

```
source_nodes = {'A','A','B'};
target_nodes = {'B','C','C'};
edge_weights = [1 2 3];
G = graph(source_nodes, target_nodes, edge_weights);
```

graph 和 **digraph** 都允许使用边列表构造简单图或多重图。构建图 G 后，可以使用命令 **G.Edges** 查看边（及其属性）。这些边在 **G.Edges** 中的顺序首先按源节点（第一列）排列，其次按目标节点（第二列）排列。对于无向图，索引较小的节点列为源节点，索引较大的节点列为目标节点。

由于 **graph** 和 **digraph** 的底层实现取决于稀疏矩阵，因此许多相同的索引创建成本均适用。使用前述方法之一基于三元对组 (**source,target,weight**) 一次性构建图比先创建空图再以迭代方式添加更多节点和

边要快。为获得最佳性能,请尽量减少对 **graph**、**digraph**、**addedge**、**addnode**、**rmedge** 和 **rmnode** 的调用次数。

图节点 ID

默认情况下,系统会对使用 **graph** 或 **digraph** 创建的图的所有节点进行编号。因此,您始终可以通过数值节点索引来引用它们。

如果图具有节点名称(即 **G.Nodes** 包含变量 **Name**),则您还可以使用节点名称来表示图中的节点。因此,可以通过节点索引或节点名称来表示图中的已命名节点。例如,可以调用节点 1: 'A'。

术语节点 ID 同时包含节点标识的两个方面的内容。节点 ID 既表示节点索引,也表示节点名称。

为方便起见,MATLAB 会记住您在调用大多数图函数时使用的节点 ID 的类型。因此,如果您使用节点索引表示图中的节点,则大多数图函数返回的数值答案也会通过节点索引来表示这些节点。

```
A = [0 1 1 0; 1 0 1 0; 1 1 0 1; 0 0 1 0];
G = graph(A,{'a','b','c','d'});
p = shortestpath(G,1,4)

p =
    1     3     4
```

但是,如果使用节点名称来表示节点,则大多数图函数返回的答案也会通过节点名称来表示这些节点(包含在字符串元胞数组或字符串数组中)。

```
p1 = shortestpath(G,'a','d')

p1 =
1×3 cell array
{'a'}  {'c'}  {'d'}
```

使用 **findnode** 查找给定的节点名称的数值节点 ID。反过来,对于给定的数值节点 ID,请创建指向 **G.Nodes.Name** 的索引以确定对应的节点名称。

修改或查询现有图

构造 **graph** 或 **digraph** 对象后,您可以使用各种函数来修改图结构或确定图有多少个节点或多少条边。下表列出了一些可用于修改或查询 **graph** 和 **digraph** 对象的函数。

addedge	在图中添加一条或多条边
rmedge	从图中删除一条或多条边
addnode	在图中添加一个或多个节点
rmnode	从图中删除一个或多个节点
findnode	查找图中的特定节点
findedge	查找图中的特定边
numnodes	计算图中的节点数
numedges	计算图中的边数

edgecount	指定的节点之间的边数
flipedge	反转有向图边的方向
reordernodes	置换图中的节点顺序
subgraph	提取子图

有关一些常见的图修改示例，请参阅“修改现有图的节点和边”（第 5-10 页）。

另请参阅

digraph | graph

详细信息

- “修改现有图的节点和边”（第 5-10 页）
- “添加图节点名称、边权重和其他属性”（第 5-13 页）
- “图的绘制和自定义”（第 5-17 页）

修改现有图的节点和边

此示例演示如何使用 `addedge`、`rmedge`、`addnode`、`rmnode`、`findedge`、`findnode` 及 `subgraph` 函数访问和修改 `graph` 或 `digraph` 对象中的节点和/或边。

添加节点

创建一个包含四个节点和四条边的图。`s` 和 `t` 中的对应元素用于指定每条图边的结束节点。

```
s = [1 1 1 2];
t = [2 3 4 3];
G = graph(s,t)

G =
graph with properties:

    Edges: [4x1 table]
    Nodes: [4x0 table]
```

查看图的边列表。

G.Edges

```
ans=4×1 table
    EndNodes
    _____
        1     2
        1     3
        1     4
        2     3
```

使用 `addnode` 向图中添加五个节点。该命令将添加五个节点 ID 分别为 5、6、7、8 和 9 的不相连节点。

```
G = addnode(G,5)

G =
graph with properties:

    Edges: [4x1 table]
    Nodes: [9x0 table]
```

删除节点

使用 `rmnode` 从图中删除节点 3、5 和 6。与其中一个已删除节点相连的所有边也会被删除。对图中剩余的六个节点重新进行编号，以反映新的节点数量。

```
G = rmnode(G,[3 5 6])

G =
graph with properties:

    Edges: [2x1 table]
    Nodes: [6x0 table]
```

添加边

使用 `addedge` 向 G 添加两条边。第一条边位于节点 1 和节点 5 之间，第二条边位于节点 2 和节点 5 之间。该命令将向 G.Edges 添加两个新行。

```
G = addedge(G,[1 2],[5 5])
```

```
G =
graph with properties:
```

```
Edges: [4x1 table]
Nodes: [6x0 table]
```

删除边

使用 `rmedge` 删除节点 1 和节点 3 之间的边。该命令将从 G.Edges 中删除一个行。

```
G = rmedge(G,1,3)
```

```
G =
graph with properties:
```

```
Edges: [3x1 table]
Nodes: [6x0 table]
```

确定边索引

确定节点 1 和 5 之间的边的边索引。边索引 `ei` 是 G.Edges 中的行号。

```
ei = findedge(G,1,5)
```

```
ei = 2
```

确定节点索引

在图中添加节点名称，然后确定节点 'd' 的节点索引。数值节点索引 `ni` 是 G.Nodes 中的行号。使用其他图函数（例如 `shortestpath`）时，可以同时使用 `ni` 和节点名称 'd' 来表示节点。

```
G.Nodes.Name = {'a' 'b' 'c' 'd' 'e' 'f'};
ni = findnode(G,'d')
```

```
ni = 4
```

提取子图

使用 `subgraph` 提取仅包含两个节点的图部分。

```
H = subgraph(G,[1 2])
```

```
H =
graph with properties:
```

```
Edges: [1x1 table]
Nodes: [2x1 table]
```

查看子图的边列表。

H.Edges

```
ans=table  
EndNodes
```

```
{'a'}  {'b'}
```

通过变量编辑器修改节点和边表格

图对象的节点和边信息包含在 **Nodes** 和 **Edges** 这两个属性中。这两个属性都是包含变量的表，用于说明图中的节点和边的特性。由于 **Nodes** 和 **Edges** 都是表，因此您可以使用变量编辑器以交互方式查看或编辑这些表。您不能使用变量编辑器添加或删除节点或边，也不能编辑 **Edges** 表的 **EndNodes** 属性。变量编辑器适用于管理 **Nodes** 和 **Edges** 表中的额外节点和边属性。有关详细信息，请参阅“[创建和编辑变量](#)”。

另请参阅

[addedge](#) | [addnode](#) | [digraph](#) | [findedge](#) | [findnode](#) | [graph](#) | [rmedge](#) | [rmnode](#) | [subgraph](#)

详细信息

- “[有向图和无向图](#)” (第 5-2 页)

添加图节点名称、边权重和其他属性

此示例演示如何向使用 `graph` 和 `digraph` 创建的图中的节点和边添加属性。当您最初调用 `graph` 或 `digraph` 来创建图时，可以指定节点名称或边权重。但是，此示例演示了如何在创建图后向图添加属性。

创建图

创建一个有向图。`s` 和 `t` 中的对应元素用于定义图中每条边的源节点和目标节点。

```
s = [1 1 2 2 3];
t = [2 4 3 4 4];
G = digraph(s,t)
```

```
G =
digraph with properties:
```

```
Edges: [5x1 table]
Nodes: [4x0 table]
```

添加节点名称

通过将变量 `Name` 添加到 `G.Nodes` 表中来向图中添加节点名称。`Name` 变量必须指定为 $N \times 1$ 字符向量元胞数组或字符串数组，其中 $N = \text{numnodes}(G)$ 。添加节点名称时请务必使用 `Name` 变量，因为该变量名称会被一些图函数进行特殊处理。

```
G.Nodes.Name = {'First' 'Second' 'Third' 'Fourth'};
```

查看新的 `Nodes` 表。

`G.Nodes`

```
ans=4×1 table
  Name
  _____
  {'First'}
  {'Second'}
  {'Third'}
  {'Fourth'}
```

使用表索引查看节点 1 和 4 的名称。

```
G.Nodes.Name([1 4])
```

```
ans = 2x1 cell
  {'First'}
  {'Fourth'}
```

添加边权重

通过将变量 `Weight` 添加到 `G.Edges` 表中来向图添加边权重。`Weight` 变量必须是 $M \times 1$ 数值向量，其中 $M = \text{numedges}(G)$ 。添加边权重时请务必使用 `Weight` 变量，因为该变量名称会被一些图函数进行特殊处理。

```
G.Edges.Weight = [10 20 30 40 50]';
```

查看新的 **Edges** 表。

G.Edges

```
ans=5×2 table
    EndNodes      Weight
    _____
    {'First'}  {'Second'}  10
    {'First'}  {'Fourth'}  20
    {'Second'} {'Third'}   30
    {'Second'} {'Fourth'}  40
    {'Third'}  {'Fourth'}  50
```

使用表索引查看 **G.Edges** 的第一行和第三行。

G.Edges([1 3],:)

```
ans=2×2 table
    EndNodes      Weight
    _____
    {'First'}  {'Second'}  10
    {'Second'} {'Third'}   30
```

添加自定义属性

原则上，您可以将任何变量添加到 **G.Nodes** 和 **G.Edges** 中，来定义图节点或边的属性。添加自定义属性很有用，因为 **subgraph** 和 **reordernodes** 之类的函数可以保留图属性。

例如，可以向 **G.Edges** 添加名为 **Power** 的变量，来指示每条边是 '**on**' 还是 '**off**'。

```
G.Edges.Power = {'on' 'on' 'on' 'off' 'off'};
G.Edges
```

```
ans=5×3 table
    EndNodes      Weight  Power
    _____
    {'First'}  {'Second'}  10  {'on'}
    {'First'}  {'Fourth'}  20  {'on'}
    {'Second'} {'Third'}   30  {'on'}
    {'Second'} {'Fourth'}  40  {'off'}
    {'Third'}  {'Fourth'}  50  {'off'}
```

向 **G.Nodes** 添加名为 **Size** 的变量，来指示每个节点的实际大小。

```
G.Nodes.Size = [10 20 10 30]';
G.Nodes
```

```
ans=4×2 table
    Name      Size
    _____
    {'First'}  10
    {'Second'} 20
```

```
{'Third' } 10
{'Fourth'} 30
```

使用变量编辑器修改表

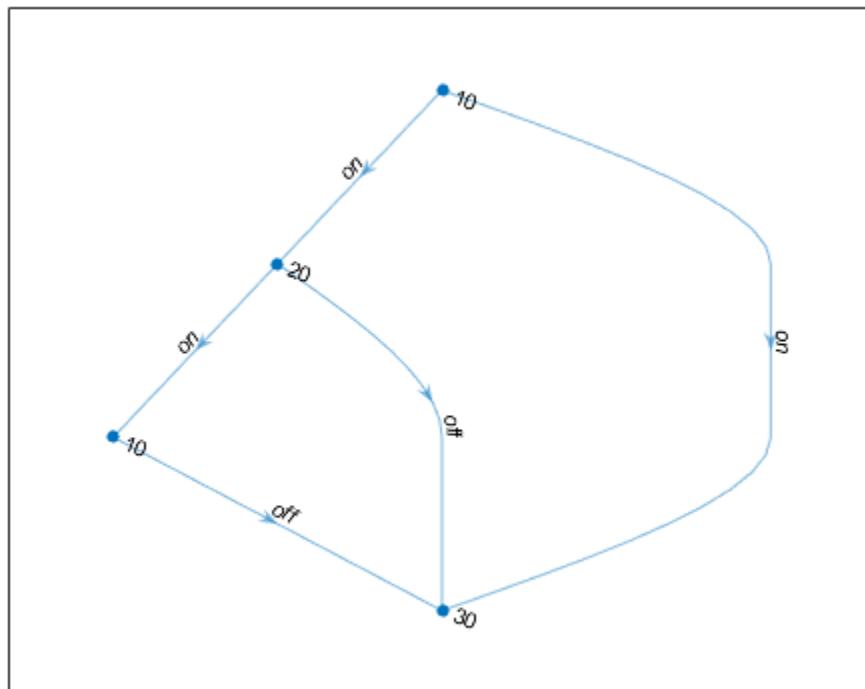
由于 `Nodes` 和 `Edges` 都是表，因此您可以使用变量编辑器以交互方式查看或编辑这些表。有关详细信息，请参阅“[创建和编辑变量](#)”。

为图论图的节点和边添加标签

绘制图时，您可以使用 `G.Nodes` 和 `G.Edges` 中的变量为图节点和边添加标签。这种做法很方便，因为已经确保这些变量具有正确数量的元素。

绘制图，并使用 `G.Edges` 中的 `Power` 变量为边添加标签。使用 `G.Nodes` 中的 `Size` 变量为节点添加标签。

```
p = plot(G,'EdgeLabel',G.Edges.Power,'NodeLabel',G.Nodes.Size)
```



```
p =
GraphPlot with properties:
```

```
NodeColor: [0 0.4470 0.7410]
MarkerSize: 4
Marker: 'o'
EdgeColor: [0 0.4470 0.7410]
LineWidth: 0.5000
LineStyle: '-'
```

```
NodeLabel: {'10' '20' '10' '30'}  
EdgeLabel: {'on' 'on' 'on' 'off' 'off'}  
XData: [2 1.5000 1 2]  
YData: [4 3 2 1]  
ZData: [0 0 0 0]
```

Show all properties

另请参阅

`digraph | graph`

详细信息

- “有向图和无向图” (第 5-2 页)
- “修改现有图的节点和边” (第 5-10 页)

图的绘制和自定义

此示例演示如何绘制图，然后自定义显示内容以向图节点和边添加标签或高亮显示。

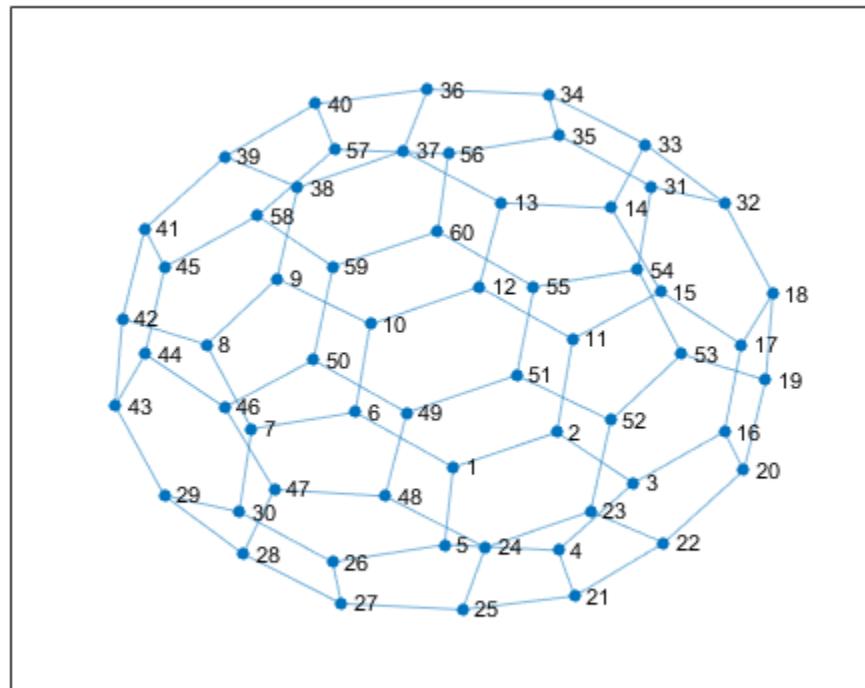
图绘图对象

使用 `plot` 函数绘制 `graph` 和 `digraph` 对象。默认情况下，`plot` 会检查图的大小和类型，以确定要使用的布局。生成的图窗窗口不包含轴刻度线。但是，如果使用 `XData`、`YData` 或 `ZData` 名称-值对组指定节点的 (x,y) 坐标，图窗将包含轴刻度线。

节点数不超过 100 的图会自动包含节点标签。节点标签使用节点名称（如果可用）；否则标签为数值节点索引。

例如，使用巴基球邻接矩阵创建一个图，然后使用所有的默认选项绘制该图。如果您调用 `plot` 并指定输出参数，则此函数将返回 `GraphPlot` 对象的句柄。随后，您可以使用该对象调整绘图的属性。例如，可以更改边的颜色或样式、节点的大小和颜色等。

```
G = graph(bucky);
p = plot(G)
```



```
p =
GraphPlot with properties:
```

`NodeColor: [0 0.4470 0.7410]`

`MarkerSize: 4`

`Marker: 'o'`

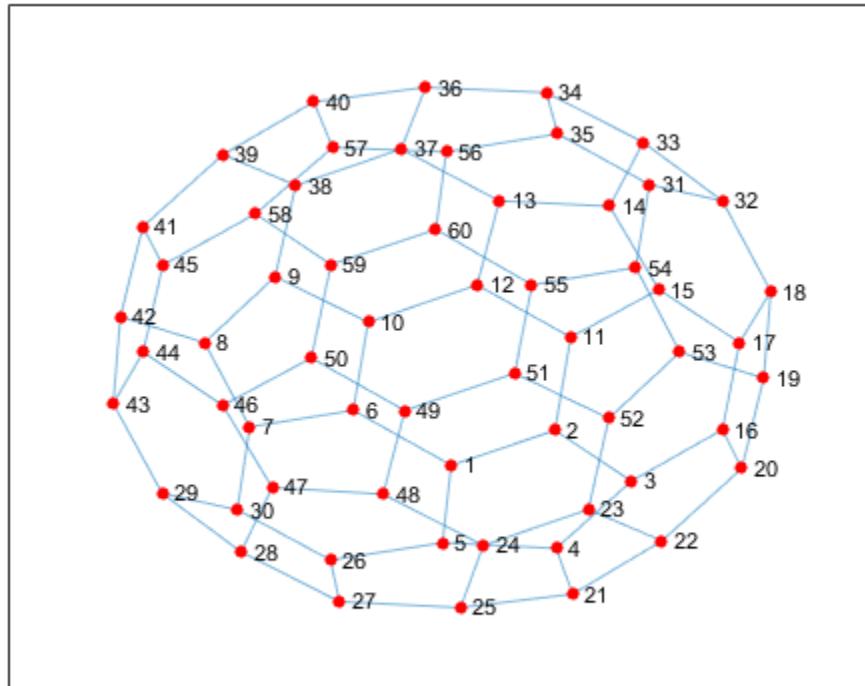
```
EdgeColor: [0 0.4470 0.7410]
LineWidth: 0.5000
LineStyle: '-'
NodeLabel: {1x60 cell}
EdgeLabel: {}
XData: [1x60 double]
YData: [1x60 double]
ZData: [1x60 double]
```

Show all properties

获得 **GraphPlot** 对象的句柄后，便可以使用点索引访问或更改属性值。有关您可以调整的属性的完整列表，请参阅 **GraphPlot** 属性。

将 **NodeColor** 的值更改为 'red'。

```
p.NodeColor = 'red';
```



确定边的线宽。

```
p.LineWidth
```

```
ans = 0.5000
```

创建并绘制图

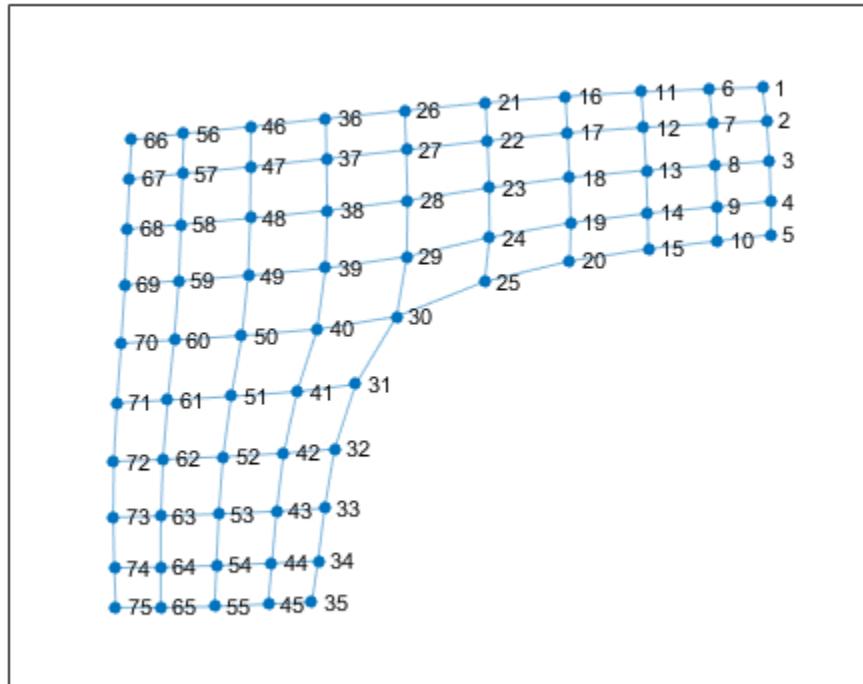
创建并绘制一个表示 L 形膜的图，L 形膜是基于一侧有 12 个节点的方形网格构建的。使用 **plot** 指定输出参数以返回 **GraphPlot** 对象的句柄。

```
n = 12;
A = delsq(numgrid('L',n));
G = graph(A,'omitselfloops')
```

```
G =
graph with properties:
```

```
Edges: [130x2 table]
Nodes: [75x0 table]
```

```
p = plot(G)
```



```
p =
GraphPlot with properties:
```

```
NodeColor: [0 0.4470 0.7410]
MarkerSize: 4
Marker: 'o'
EdgeColor: [0 0.4470 0.7410]
LineWidth: 0.5000
LineStyle: '-'
NodeLabel: {1x75 cell}
```

```
EdgeLabel: {}
XData: [1x75 double]
YData: [1x75 double]
ZData: [1x75 double]
```

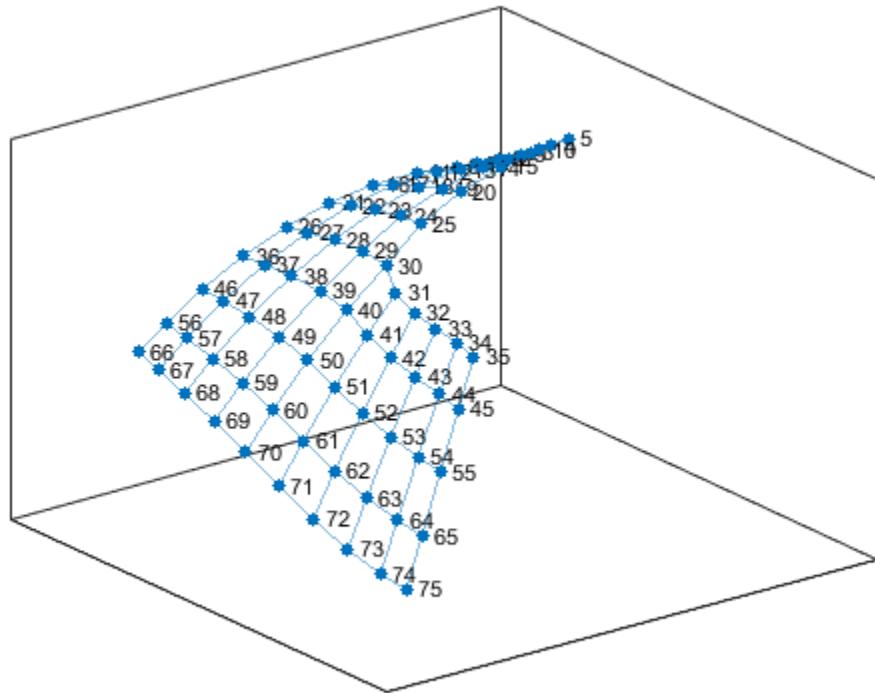
Show all properties

更改图节点的布局

使用 `layout` 函数更改绘图中的图节点的布局。不同的布局选项会自动计算绘图的节点坐标。或者，可以使用 `GraphPlot` 对象的 `XData`、`YData` 和 `ZData` 属性来指定您自己的节点坐标。

不使用默认的二维布局方法，而是使用 `layout` 来指定 '`force3`' 布局（三维力导向图布局）。

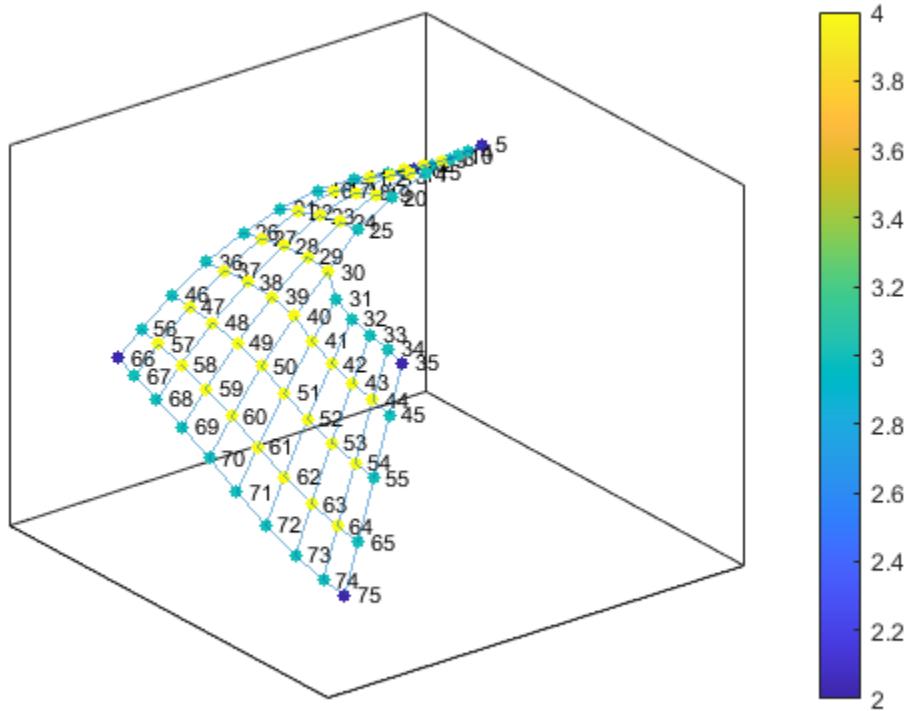
```
layout(p,'force3')
view(3)
```



按比例对节点着色

根据图节点的出入度为它们着色。在该图中，所有内部节点都具有最大度数 4，沿图边的节点的度数为 3，角节点具有最小度数 2。将该节点着色数据存储为 `G.Nodes` 中的变量 `NodeColors`。

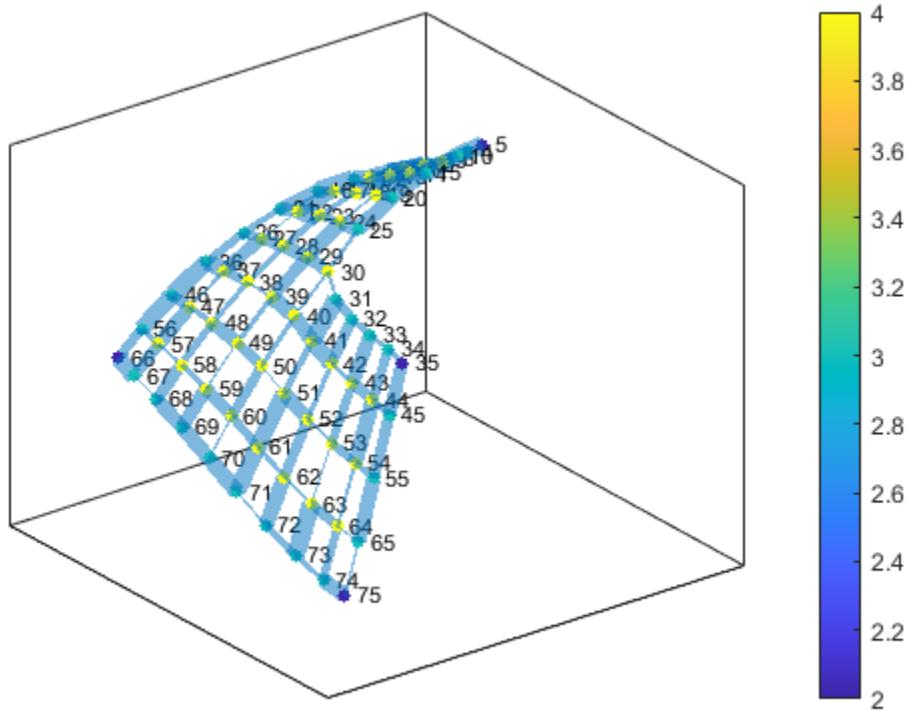
```
G.Nodes.NodeColors = degree(G);
p.NodeCData = G.Nodes.NodeColors;
colorbar
```



按权重列出的边线宽度

向图边添加一些随机整数权重，然后绘制这些边，使它们的线宽与权重成比例。由于约大于 7 的边线宽度开始变得很复杂，因此缩放线宽，使权重最大的边的线宽为 7。将该边宽数据存储为 G.Edges 中的变量 LWidths。

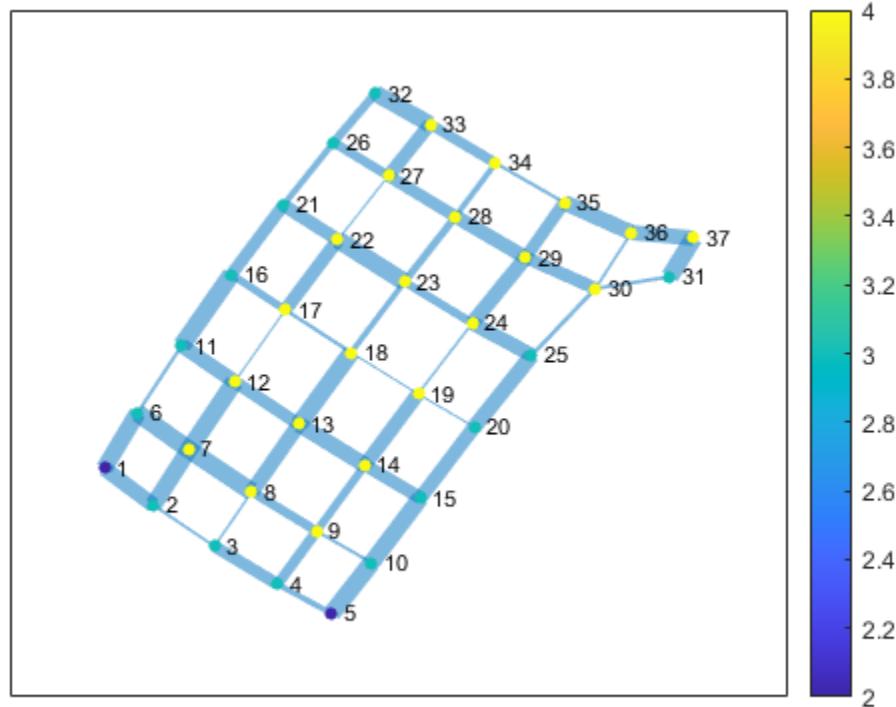
```
G.Edges.Weight = randi([10 250],130,1);
G.Edges.LWidths = 7*G.Edges.Weight/max(G.Edges.Weight);
p.LineWidth = G.Edges.LWidths;
```



提取子图

提取 G 的右上角并将其作为子图绘制，以更便于读取图上的详细信息。新图 H 从 G 继承 NodeColors 和 LWidths 变量，因此最直接的方式就是重新创建之前的绘图自定义项。但是，系统会对 H 中的节点重新进行编号，以将图中的新节点编号考虑在内。

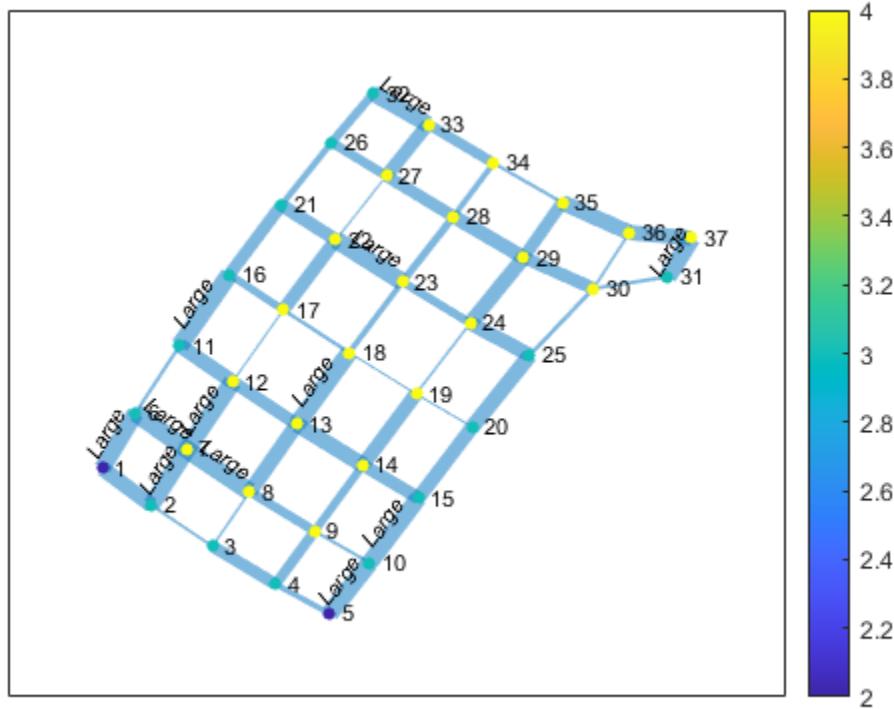
```
H = subgraph(G,[1:31 36:41]);
p1 = plot(H,'NodeCData',H.Nodes.NodeColors,'LineWidth',H.Edges.LWidths);
colorbar
```



为节点和边添加标签

使用 `labeledge` 对宽度大于 6 的边添加标签 'Large'。`labelnode` 函数以相似的方式为节点添加标签。

```
labeledge(p1,find(H.Edges.LWidths > 6),'Large')
```

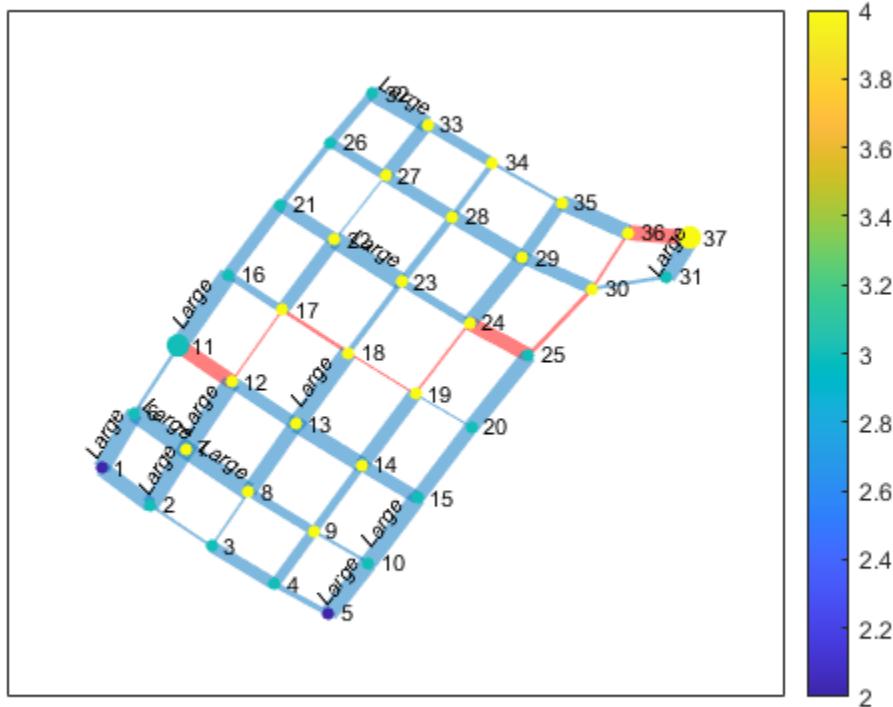


突出显示最短路径

查找子图 H 中节点 11 与节点 37 之间的最短路径。以红色高亮显示沿此路径的边，并增大路径的结束节点的大小。

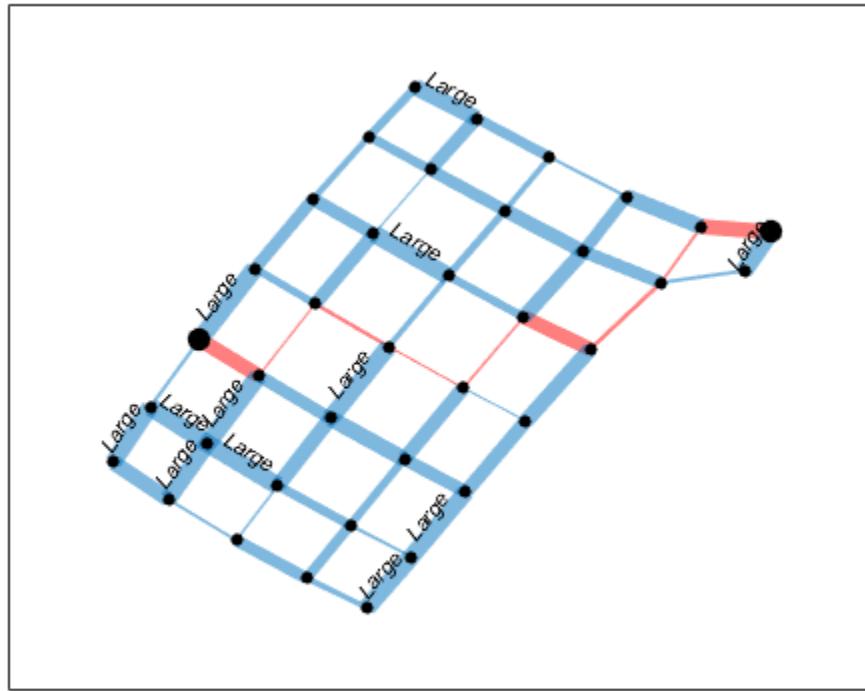
```
path = shortestpath(H,11,37)
path = 1×10
    11    12    17    18    19    24    25    30    36    37

highlight(p1,[11 37])
highlight(p1,path,'EdgeColor','r')
```



删除节点标签和颜色栏，并使所有节点都变成黑色。

```
p1.NodeLabel = {};
colorbar off
p1.NodeColor = 'black';
```



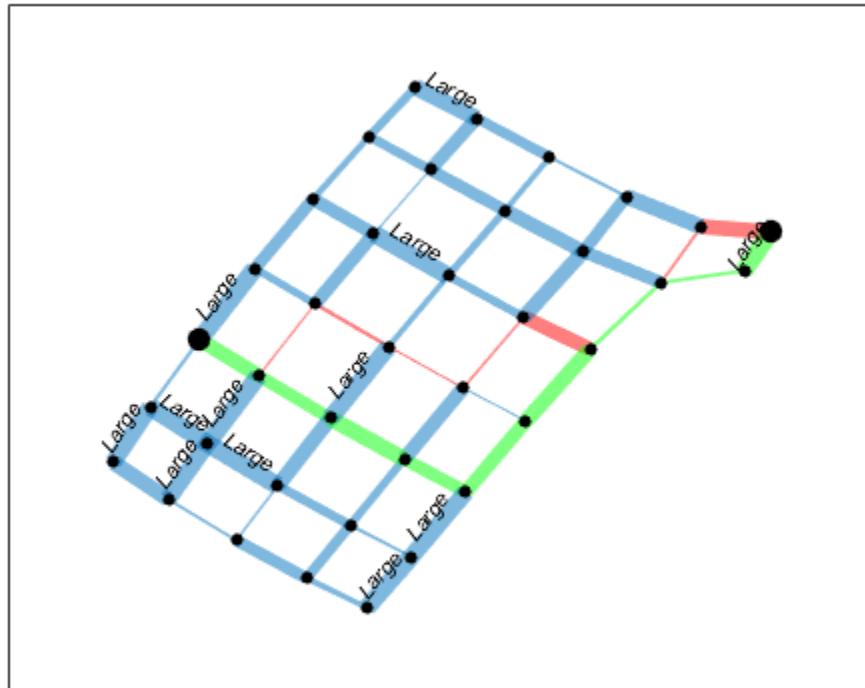
查找忽略边权重的其他最短路径。以绿色突出显示此路径。

```
path2 = shortestpath(H,11,37,'Method','unweighted')
```

```
path2 = 1×10
```

```
11 12 13 14 15 20 25 30 31 37
```

```
highlight(p1,path2,'EdgeColor','g')
```



绘制大图

创建包含数十万个甚至数百万个节点和/或边的图是很常见的。为此，**plot** 处理大图会略有不同，以保持可读性和性能。处理节点超过 100 个的图时，**plot** 函数会进行以下调整：

- 1 默认的图布局方法始终为 '`'subspace'`'。
- 2 不再自动为这些节点添加标签。
- 3 `MarkerSize` 属性设置为 2。 (较小的图的标记大小为 4)。
- 4 有向图的 `ArrowSize` 属性设置为 4。 (较小的有向图使用的箭头大小为 7)。

另请参阅

[GraphPlot](#) | [digraph](#) | [graph](#) | [plot](#)

详细信息

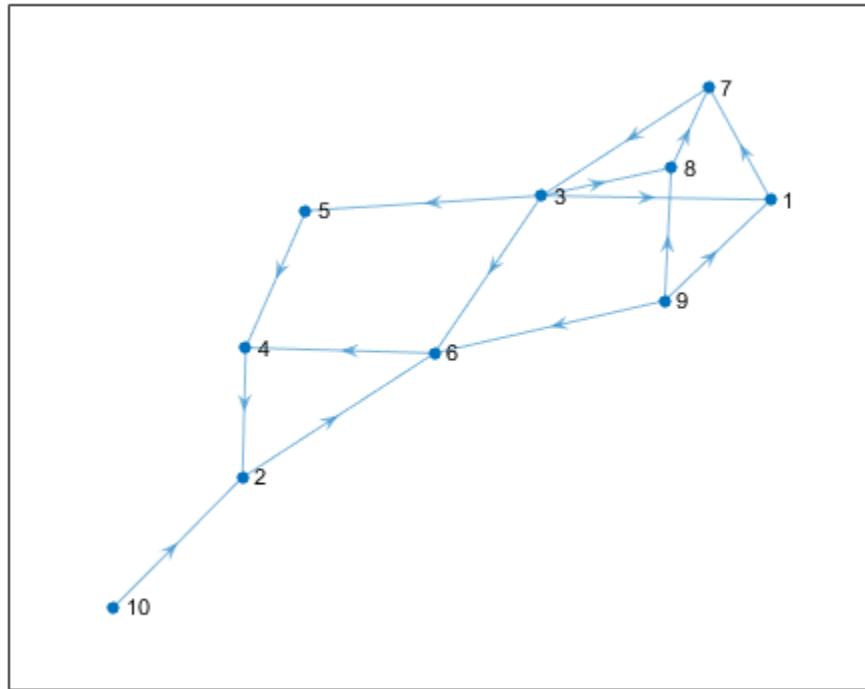
- “有向图和无向图” (第 5-2 页)
- [GraphPlot](#)
- “将节点属性添加到图论图数据游标” (第 5-36 页)

可视化广度优先搜索和深度优先搜索

此示例说明如何定义这样的函数：该函数通过突出显示图的节点和边来显示 `bfsearch` 和 `dfsearch` 的可视化结果。

创建并绘制一个有向图。

```
s = [1 2 3 3 3 4 5 6 7 8 9 9 9 10];
t = [7 6 1 5 6 8 2 4 4 3 7 1 6 8 2];
G = digraph(s,t);
plot(G)
```



对该图执行深度优先搜索。指定 '`allevents`' 以便在算法中返回所有事件。此外，将 `Restart` 指定为 `true` 以确保搜索会访问图中的每个节点。

```
T = dfsearch(G, 1, 'allevents', 'Restart', true)
```

`T =`

38x4 table

Event	Node	Edge	EdgeIndex
startnode	1	NaN	NaN
discovernode	1	NaN	NaN

```

edgetonew      NaN   1   7   1
discovernode   7   NaN   NaN   NaN
edgetonew      NaN   7   3   10
discovernode   3   NaN   NaN   NaN
edgetodiscovered NaN   3   1   3
edgetonew      NaN   3   5   4
discovernode   5   NaN   NaN   NaN
edgetonew      NaN   5   4   8
discovernode   4   NaN   NaN   NaN
edgetonew      NaN   4   2   7
discovernode   2   NaN   NaN   NaN
edgetonew      NaN   2   6   2
discovernode   6   NaN   NaN   NaN
edgetodiscovered NaN   6   4   9
finishnode    6   NaN   NaN   NaN
finishnode    2   NaN   NaN   NaN
finishnode    4   NaN   NaN   NaN
finishnode    5   NaN   NaN   NaN
edgetofinished  NaN   3   6   5
edgetonew      NaN   3   8   6
discovernode   8   NaN   NaN   NaN
edgetodiscovered NaN   8   7   11
finishnode    8   NaN   NaN   NaN
finishnode    3   NaN   NaN   NaN
finishnode    7   NaN   NaN   NaN
finishnode    1   NaN   NaN   NaN
startnode     9   NaN   NaN   NaN
discovernode   9   NaN   NaN   NaN
edgetofinished  NaN   9   1   12
edgetofinished  NaN   9   6   13
edgetofinished  NaN   9   8   14
finishnode    9   NaN   NaN   NaN
startnode     10  NaN  NaN   NaN
discovernode   10  NaN  NaN   NaN
edgetofinished  NaN  10  2   15
finishnode    10  NaN  NaN   NaN

```

表 T 中的值对可视化搜索很有用。函数 `visualize_search.m` 展示了一种方法，使用通过 `bfsearch` 和 `dfsearch` 执行的搜索的结果，根据事件表 T 突出显示图中的节点和边。该函数在算法中的每一步执行前都会暂停，这样您就可以通过按任意键缓慢地逐步执行搜索。

将 `visualize_search.m` 保存在当前文件夹中。

```

function visualize_search(G,t)
% G is a graph or digraph object, and t is a table resulting from a call to
% BFSEARCH or DFSEARCH on that graph.
%
% Example inputs: G = digraph([1 2 3 3 3 4 5 6 7 8 9 9 10], ...
% [7 6 1 5 6 8 2 4 4 3 7 1 6 8 2]);
% t = dfsearch(G, 1, 'allevents', 'Restart', true);

% Copyright 1984-2019 The MathWorks, Inc.

isundirected = isa(G, 'graph');
if isundirected
    % Replace graph with corresponding digraph, because we need separate

```

```

% edges for both directions
[src, tgt] = findedge(G);
G = digraph([src; tgt], [tgt; src], [1:numedges(G), 1:numedges(G)]);
end

h = plot(G,'NodeColor',[0.5 0.5 0.5],'EdgeColor',[0.5 0.5 0.5], ...
'EdgeLabelMode', 'auto');

for ii=1:size(t,1)
    switch t.Event(ii)
        case 'startnode'
            highlight(h,t.Node(ii),'MarkerSize',min(h.MarkerSize)*2);
        case 'discovernode'
            highlight(h,t.Node(ii),'NodeColor','r');
        case 'finishnode'
            highlight(h,t.Node(ii),'NodeColor','k');
        otherwise
            if isundirected
                a = G.Edges.Weight;
                b = t.EdgeIndex(ii);
                edgeind = intersect(find(a == b),...
                    findedge(G,t.Edge(ii,1),t.Edge(ii,2)));
            else
                edgeind = t.EdgeIndex(ii);
            end
            switch t.Event(ii)
                case 'edgetonew'
                    highlight(h,'Edges',edgeind,'EdgeColor','b');
                case 'edgetodiscovered'
                    highlight(h,'Edges',edgeind,'EdgeColor',[0.8 0 0.8]);
                case 'edgetofinished'
                    highlight(h,'Edges',edgeind,'EdgeColor',[0 0.8 0]);
            end
        end
    end

nodeStr = t.Node;
if isnumeric(nodeStr)
    nodeStr = num2cell(nodeStr);
    nodeStr = cellfun(@num2str, nodeStr, 'UniformOutput', false);
end

edgeStr = t.Edge;
if isnumeric(edgeStr)
    edgeStr = num2cell(edgeStr);
    edgeStr = cellfun(@num2str, edgeStr, 'UniformOutput', false);
end

if ~isnan(t.Node(ii))
    title([char(t{ii, 1}) ' on Node ' nodeStr{ii}]);
else
    title([char(t{ii, 1}) ' on Edge (' edgeStr{ii, 1} ',',...
        edgeStr{ii, 2},') with edge index ' sprintf('%d ', t{ii, 4})]);
end

disp('Strike any key to continue...')
pause
end
disp('Done.')

```

```
close all
```

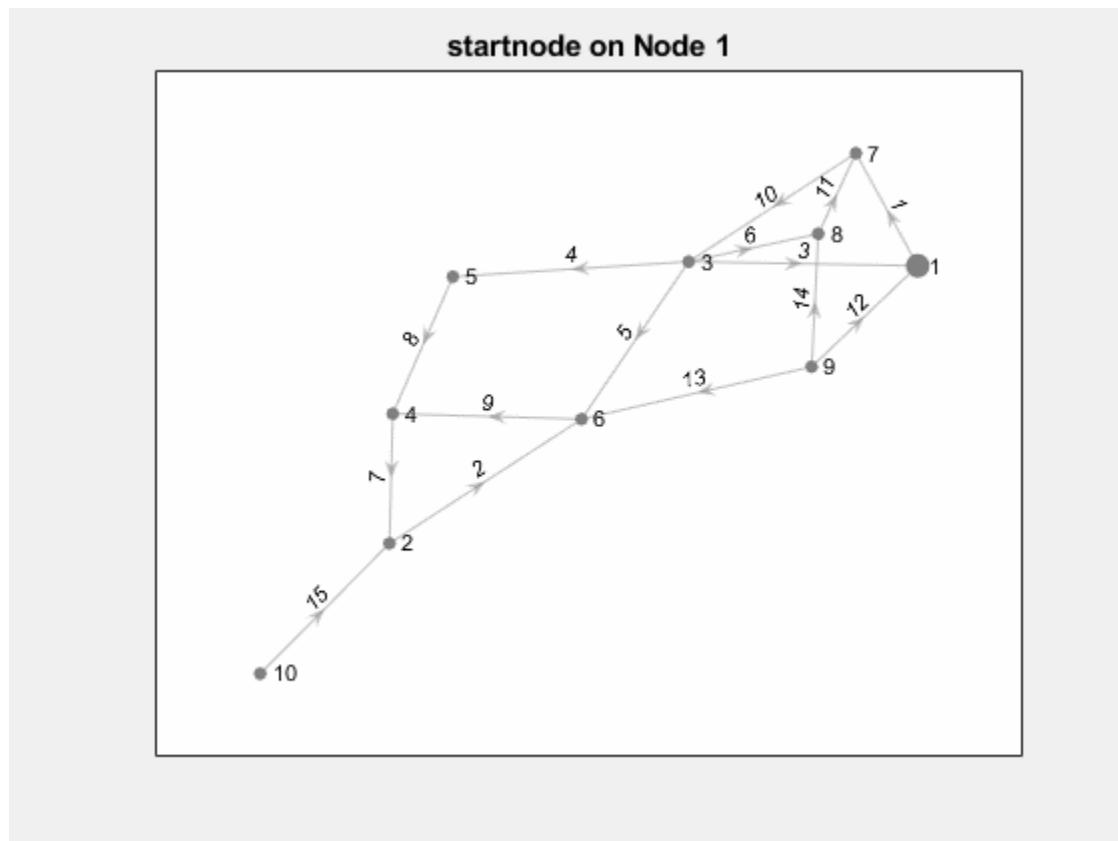
使用以下命令对图 G 和搜索结果 T 运行 `visualize_search.m`:

```
visualize_search(G,T)
```

该图一开始全为灰色，然后每次您按一个键，就会出现一条新的搜索结果。根据以下命令高亮显示搜索结果：

- 'startnode' - 起始节点的大小变大。
- 'discovernode' - 节点在被发现后变成红色。
- 'finishnode' - 节点在完成后变成黑色。
- 'edgetonew' - 通向未发现的节点的边变成蓝色。
- 'edgetodiscovered' - 通向已发现的节点的边变成品红色。
- 'edgetofinished' - 通向已完成的节点的边变成绿色。

下面的 .gif 动画展示当您步进 `visualize_search.m` 的结果时所看到的内容。



另请参阅

[bfsearch](#) | [dfsearch](#) | [digraph](#) | [graph](#)

详细信息

- “有向图和无向图” (第 5-2 页)

使用拉普拉斯矩阵为图分区

此示例说明如何使用图的拉普拉斯矩阵来计算 Fiedler 向量。Fiedler 向量可用于将图分区为两个子图。

加载数据

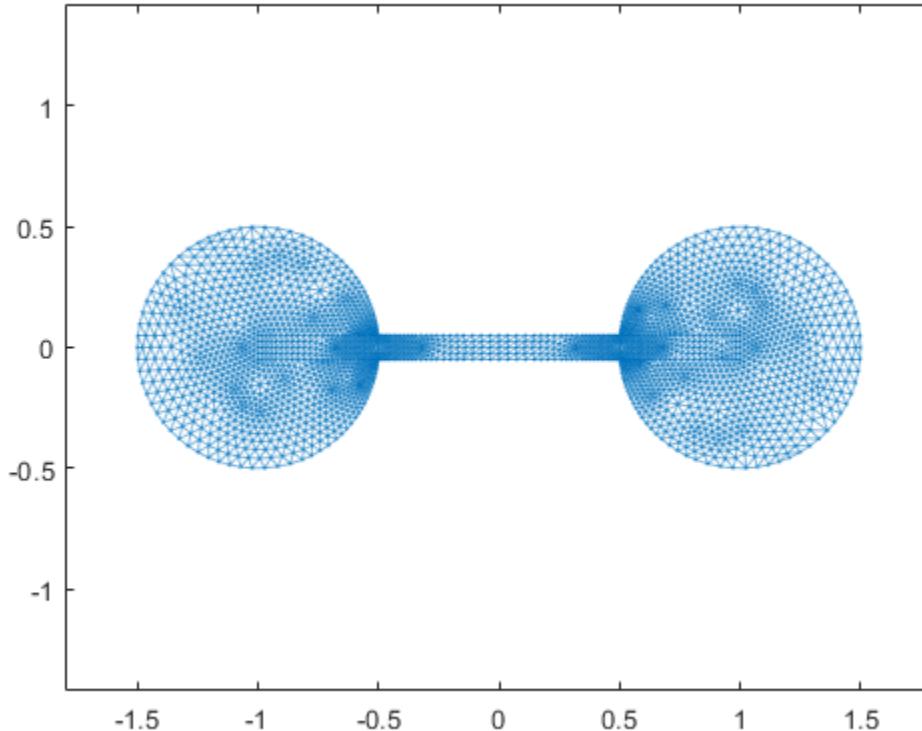
加载数据集 `barbellgraph.mat`, 其中包含一个杠铃图的稀疏邻接矩阵和节点坐标。

```
load barbellgraph.mat
```

绘制图

使用自定义节点坐标 `xy` 绘制图。

```
G = graph(A,'omitselfloops');
p = plot(G,'XData',xy(:,1),'YData',xy(:,2),'Marker','.');
axis equal
```



计算拉普拉斯矩阵和 Fiedler 向量

计算图的拉普拉斯矩阵。然后，使用 `eigs` 计算两个模最小的特征值和相应的特征向量。

```
L = laplacian(G);
[V,D] = eigs(L,2,'smallestabs');
```

Fiedler 向量是对应于该图的次最小特征值的特征向量。最小特征值为零，表示该图包含一个连通分量。这种情况下，`V` 中的第二列对应于次最小特征值 $D(2,2)$ 。

D

$$\mathbf{D} = 2 \times 2 \\ 10^{-3} \times$$

$$\begin{matrix} 0.0000 & 0 \\ 0 & 0.2873 \end{matrix}$$

```
w = V(:,2);
```

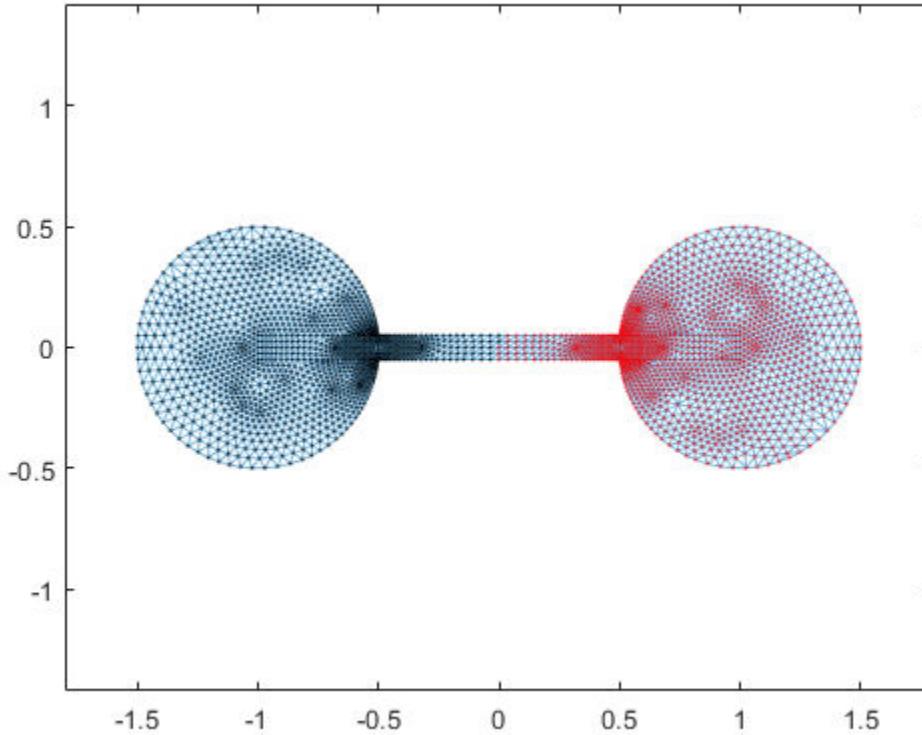
由于仅计算特征值和特征向量的子集，因此使用 `eigs` 求 Fiedler 向量的方法可扩展至更大的图，但对于较小的图而言，将拉普拉斯矩阵转换为满存储并使用 `eig(full(L))` 同样切实可行。

为图分区

使用 Fiedler 向量 \mathbf{w} 将图分区为两个子图。如果一个节点在 \mathbf{w} 中具有正值，则该节点将分配至子图 A。否则，该节点将分配至子图 B。这种做法称为符号切割或零阈值切割。符号切割最大限度减小了切割权重，该权重受限于图的任意非平凡切割的权重上界和下界。

使用符号切割对图进行分区。将 $\mathbf{w} \geq 0$ 的节点的子图突出显示为红色，将 $\mathbf{w} < 0$ 的节点的子图突出显示为黑色。

```
highlight(p,find(w>=0),'NodeColor','r') % subgraph A  
highlight(p,find(w<0),'NodeColor','k') % subgraph B
```



对于该杠铃图而言，此分区恰好将图平分为两个相等的节点集。但符号切割不总是生成平衡切割。

任何时候均可通过计算 w 的中位数并将其用作阈值来平分图。这种分区方法被称为中位数切割，它能保证每个子图具有相等的节点数量。

使用中位数切割时，首先按中位数平移 w 中的值：

```
w_med = w - median(w);
```

然后，按 w_{med} 中的符号对图进行分区。对杠铃图而言， w 的中位数接近于零，因此两次切割会生成相似的平分。

另请参阅

`digraph` | `graph` | `laplacian` | `subgraph`

详细信息

- “有向图和无向图”（第 5-2 页）

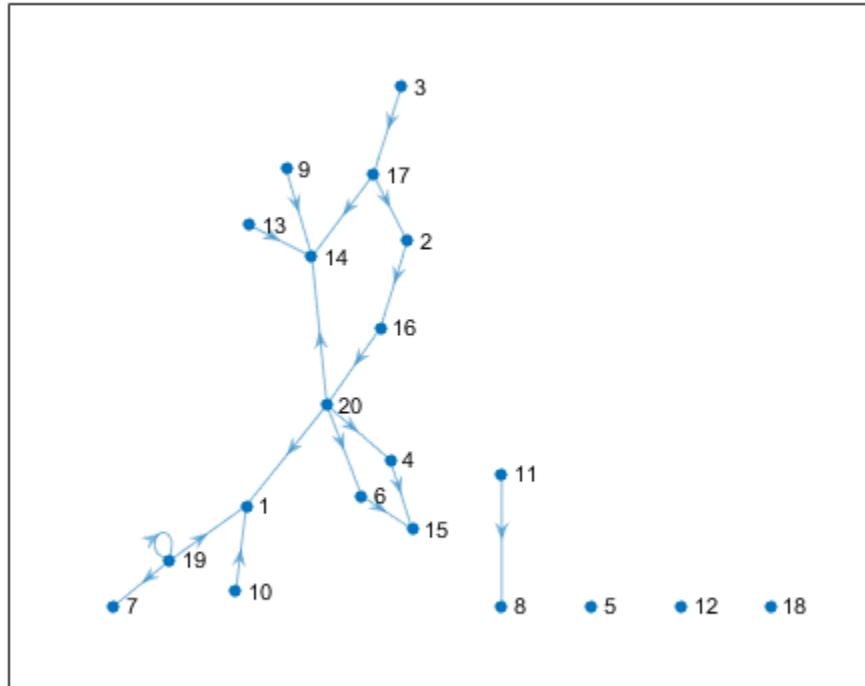
将节点属性添加到图论图数据游标

此示例说明如何自定义 GraphPlot 数据游标，以显示图的额外节点属性。

创建 GraphPlot 对象

创建随机有向图的 GraphPlot 图对象。将额外的节点属性 `wifi` 添加到该图。

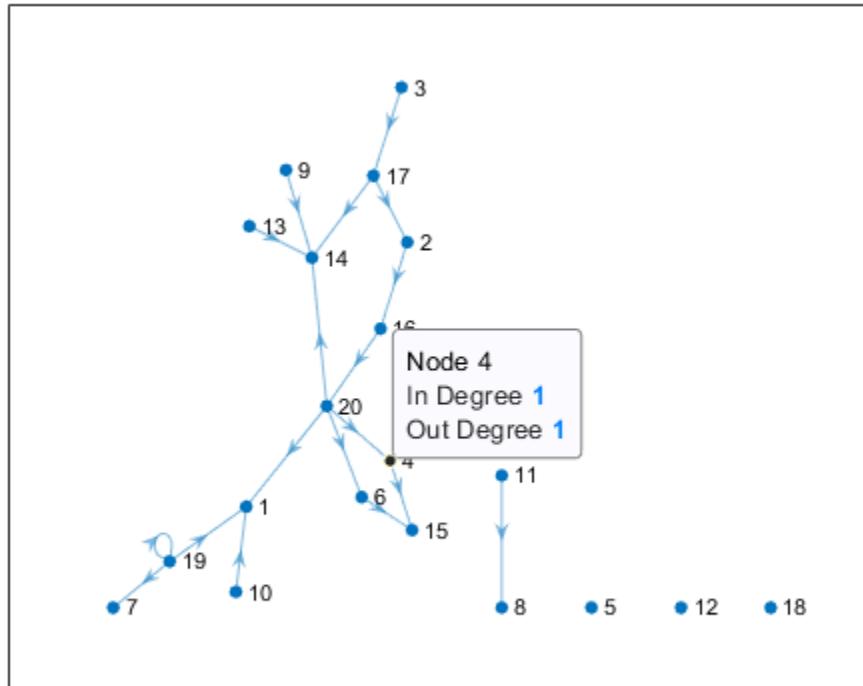
```
rng default
G = digraph(sprandn(20, 20, 0.05));
G.Nodes.wifi = randi([0 1], 20, 1) == 1;
h = plot(G);
```



启用数据游标模式

通过从工具菜单中选择**数据游标**菜单项，启用数据游标模式。

启用数据游标模式后，点击图中的节点以调用游标显示。利用数据游标，您可以选择图论图中的节点并查看节点的属性。默认情况下，无向图的数据游标会显示节点 ID 编号和度。对于有向图，显示内容包括节点 ID 编号、入度和出度。



要显示额外的节点属性（例如 wifi），需要修改数据游标回调函数。有关使用和自定义数据游标的一般信息，请参阅“交互式探查绘图数据”。

自定义数据游标所显示的文本

通过编写新的数据游标回调函数，自定义数据游标所显示的文本。

1. 在您的当前目录中保存函数 **GraphCursorCallback.m**:

```
function output_txt = GraphCursorCallback(obj,event_obj,NodeProperties)
% Display the position of the data cursor
% obj      Currently not used (empty)
% event_obj Handle to event object
% output_txt Data cursor text (character vector or cell array of character vectors).

h = get(event_obj,'Target');
pos = get(event_obj,'Position');
ind = find(h.XData == pos(1) & h.YData == pos(2), 1);

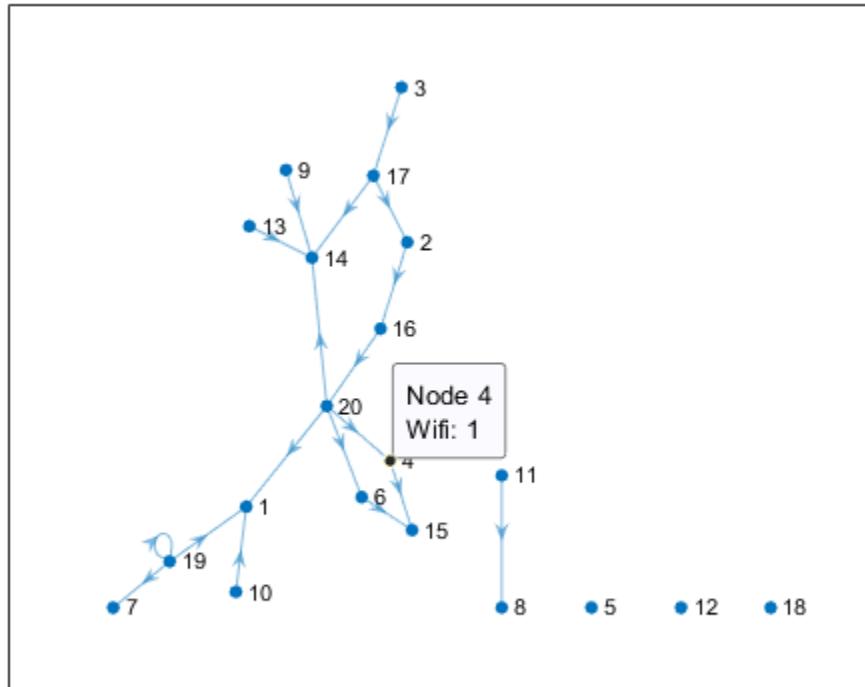
output_txt = {[ 'Node ' num2str(ind)], ...
[ 'Wifi: ' num2str(NodeProperties.wifi(ind))]};
```

标准数据游标回调函数可接受两个输入参数。**GraphCursorCallback** 可接受一个额外的输入参数 **NodeProperties**，使数据游标能够获得图中的额外节点属性（例如 wifi）的访问权。

2. 通过更改数据游标管理器对象的 `UpdateFcn` 属性，将 `GraphCursorCallback` 函数连接到数据游标。此命令使用匿名函数，将 `G.Nodes` 表作为第三个输入传递到 `GraphCursorCallback`。此方法仅获取 `G.Nodes` 表的快照，因此，如果图属性后续发生更改，则您需要再次更改 `UpdateFcn` 属性。

```
hdt = datacursormode;
hdt.UpdateFcn = @(obj,event_obj) GraphCursorCallback(obj,event_obj,G.Nodes);
```

3. 现在 `GraphCursorCallback` 已连接到数据游标，删除绘图中的任何原有数据提示，然后重新启用数据游标模式并选择一个节点。新的数据游标显示将包含在 `GraphCursorCallback` 中定义的 `wifi` 节点属性。



另请参阅

`datacursormode` | `digraph` | `graph`

详细信息

- “交互式探查绘图数据”

生成 Watts-Strogatz 小世界图形模型

此示例演示如何构建并分析 Watts-Strogatz 小世界图形。Watts-Strogatz 模型是一个包含小世界网络属性（例如群集和平均短路径长度）的随机图形。

算法说明

创建 Watts-Strogatz 图形包含以下两个基本步骤：

- 1 创建一个环形网格，其中包含 N 个平均出入度为 $2K$ 的节点。每个节点都与任一侧的 K 个最近邻点相连。
- 2 对于图中的每条边，重新连接概率为 β 的目标节点。重新连接的边不能是重复或自环的。

执行第一步操作之后，图形将是一个完美的环形网格。因此当 $\beta = 0$ 时，不会重新连接任何边，并且该模型会返回一个环形网格。而如果 $\beta = 1$ ，则所有边都将重新连接，并且环形网格会变成随机图形。

文件 `WattsStrogatz.m` 对无向图实现该图形算法。根据上面的算法说明，输入参数为 N 、 K 和 beta 。

查看文件 `WattsStrogatz.m`。

```
% Copyright 2015 The MathWorks, Inc.

function h = WattsStrogatz(N,K,beta)
% H = WattsStrogatz(N,K,beta) returns a Watts-Strogatz model graph with N
% nodes, N*K edges, mean node degree 2*K, and rewiring probability beta.
%
% beta = 0 is a ring lattice, and beta = 1 is a random graph.

% Connect each node to its K next and previous neighbors. This constructs
% indices for a ring lattice.
s = repelem((1:N)',1,K);
t = s + repmat(1:K,N,1);
t = mod(t-1,N)+1;

% Rewire the target node of each edge with probability beta
for source=1:N
    switchEdge = rand(K, 1) < beta;

    newTargets = rand(N, 1);
    newTargets(source) = 0;
    newTargets(s(t==source)) = 0;
    newTargets(t(source, ~switchEdge)) = 0;

    [~, ind] = sort(newTargets, 'descend');
    t(source, switchEdge) = ind(1:nnz(switchEdge));
end

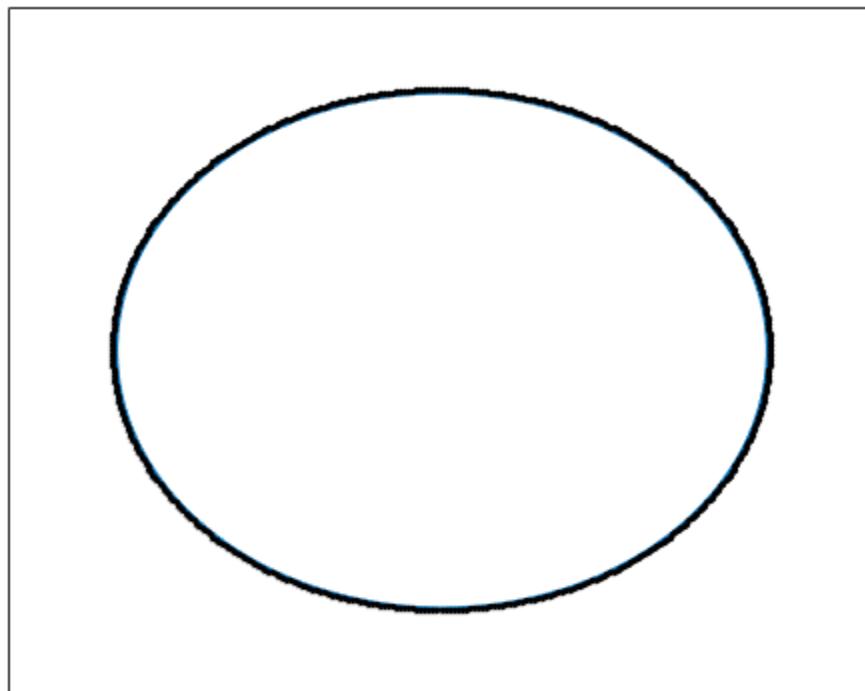
h = graph(s,t);
end
```

环形网格

使用 `WattsStrogatz` 函数构建包含 500 个节点的环形网格。`beta` 为 0 时，该函数会返回其节点出入度均为 $2K$ 的环形网格。

```
h = WattsStrogatz(500,25,0);
plot(h,'NodeColor','k','Layout','circle');
title('Watts-Strogatz Graph with $N = 500$ nodes, $K = 25$, and $\beta = 0$', ...
'Interpreter','latex')
```

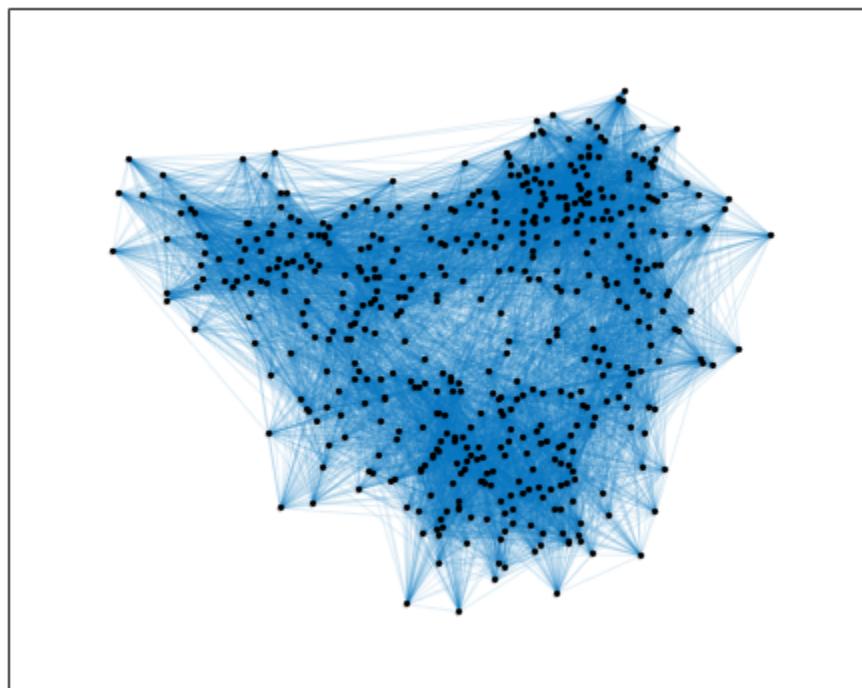
Watts-Strogatz Graph with $N = 500$ nodes, $K = 25$, and $\beta = 0$



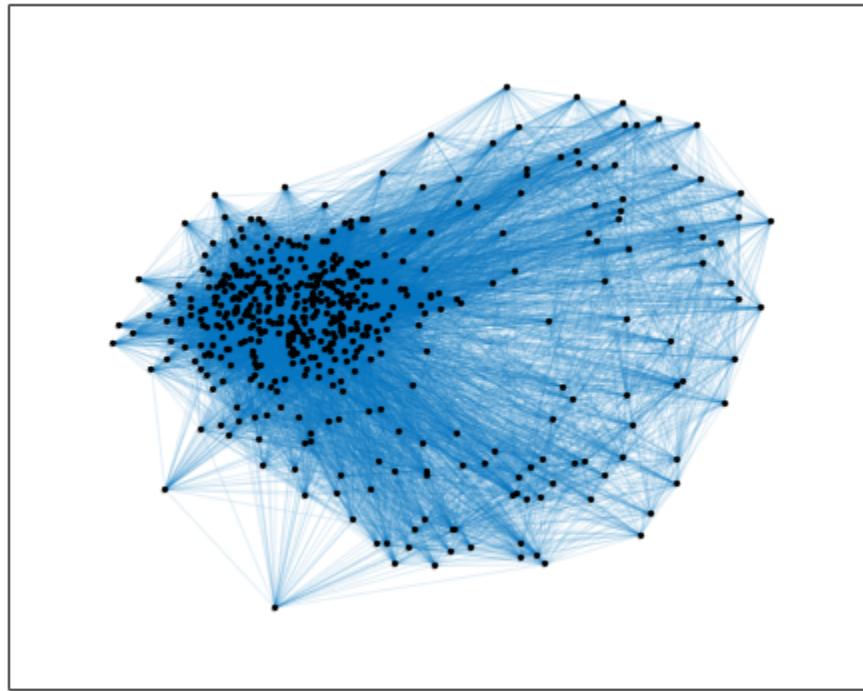
一些随机边

通过将 `beta` 增大到 0.15 和 0.50，增加图形中的随机性。

```
h2 = WattsStrogatz(500,25,0.15);
plot(h2,'NodeColor','k','EdgeAlpha',0.1);
title('Watts-Strogatz Graph with $N = 500$ nodes, $K = 25$, and $\beta = 0.15$', ...
'Interpreter','latex')
```

Watts-Strogatz Graph with $N = 500$ nodes, $K = 25$, and $\beta = 0.15$ 

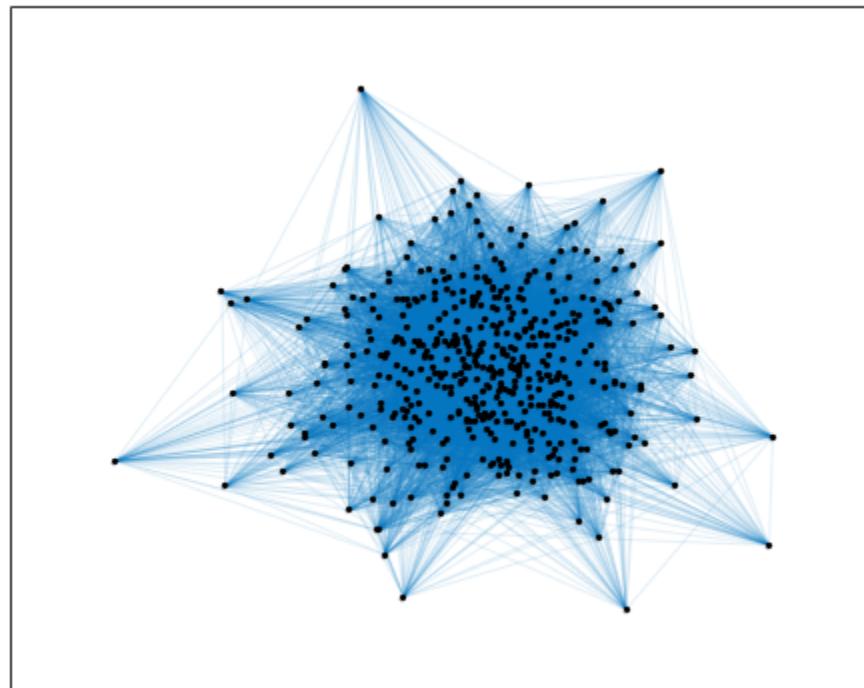
```
h3 = WattsStrogatz(500,25,0.50);
plot(h3,'NodeColor','k','EdgeAlpha',0.1);
title('Watts-Strogatz Graph with $N = 500$ nodes, $K = 25$, and $\beta = 0.50$', ...
'Interpreter','latex')
```

Watts-Strogatz Graph with $N = 500$ nodes, $K = 25$, and $\beta = 0.50$ 

随机图形

通过将 `beta` 增大到其最大值 1.0，生成一个完全随机的图形。这会重新连接所有边。

```
h4 = WattsStrogatz(500,25,1);
plot(h4,'NodeColor','k','EdgeAlpha',0.1);
title('Watts-Strogatz Graph with $N = 500$ nodes, $K = 25$, and $\beta = 1$', ...
'Interpreter','latex')
```

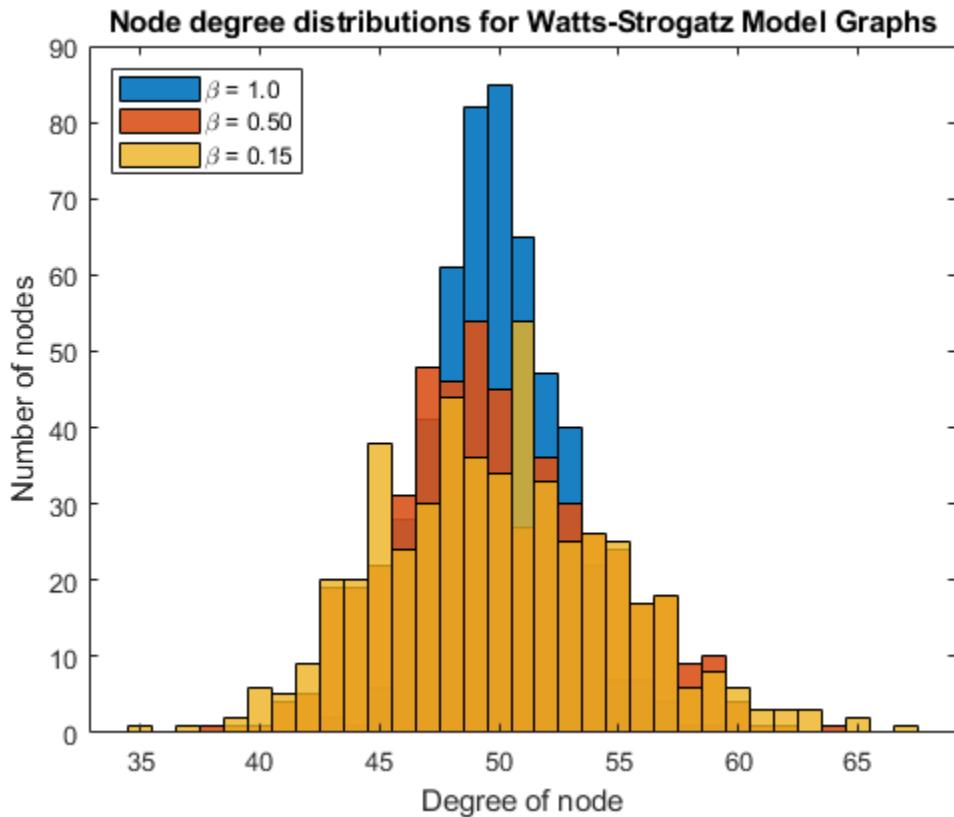
Watts-Strogatz Graph with $N = 500$ nodes, $K = 25$, and $\beta = 1$ 

出入度分布

不同 Watts-Strogatz 图形中节点的出入度分布会有所不同。`beta` 为 0 时，这些节点的出入度均为 $2K$ ，因此出入度分布只是一个以 $2K$ 为中心的 Dirac-delta 函数 $\delta(x - 2K)$ 。但是，随着 `beta` 的增大，出入度分布也会发生变化。

该绘图显示 `beta` 的非零值的出入度分布。

```
histogram(degree(h2),'BinMethod','integers','FaceAlpha',0.9);
hold on
histogram(degree(h3),'BinMethod','integers','FaceAlpha',0.9);
histogram(degree(h4),'BinMethod','integers','FaceAlpha',0.8);
hold off
title('Node degree distributions for Watts-Strogatz Model Graphs')
xlabel('Degree of node')
ylabel('Number of nodes')
legend('\beta = 1.0','\beta = 0.50','\beta = 0.15','Location','NorthWest')
```



枢纽的形成

Watts-Strogatz 图形具有较大的群集系数，因此节点往往会展开成团或较小的紧密互联的节点组。当 **beta** 朝其最大值 1.0 方向增加时，您会看到枢纽节点或度数相对较高的节点的数量不断增加。枢纽是其他节点之间以及图形中的团之间的常见连接。枢纽之所以存在，旨在允许形成团的同时保留平均短路径长度。

对 **beta** 的每个值计算平均路径长度和枢纽节点数量。就此示例而言，枢纽节点是指入度大于或等于 55 的节点。与原始环形网格相比，这些节点的出入度全都增加了 10% 或更多。

```
n = 55;
d = [mean(mean(distances(h))), nnz(degree(h)>=n); ...
      mean(mean(distances(h2))), nnz(degree(h2)>=n); ...
      mean(mean(distances(h3))), nnz(degree(h3)>=n);
      mean(mean(distances(h4))), nnz(degree(h4)>=n)];
T = table([0 0.15 0.50 1]', d(:,1), d(:,2),...
          'VariableNames',{'Beta','AvgPathLength','NumberOfHubs'})
```

T =

4x3 table

Beta	AvgPathLength	NumberOfHubs
0	5.48	0
0.15	2.0715	20

0	5.48	0
0.15	2.0715	20

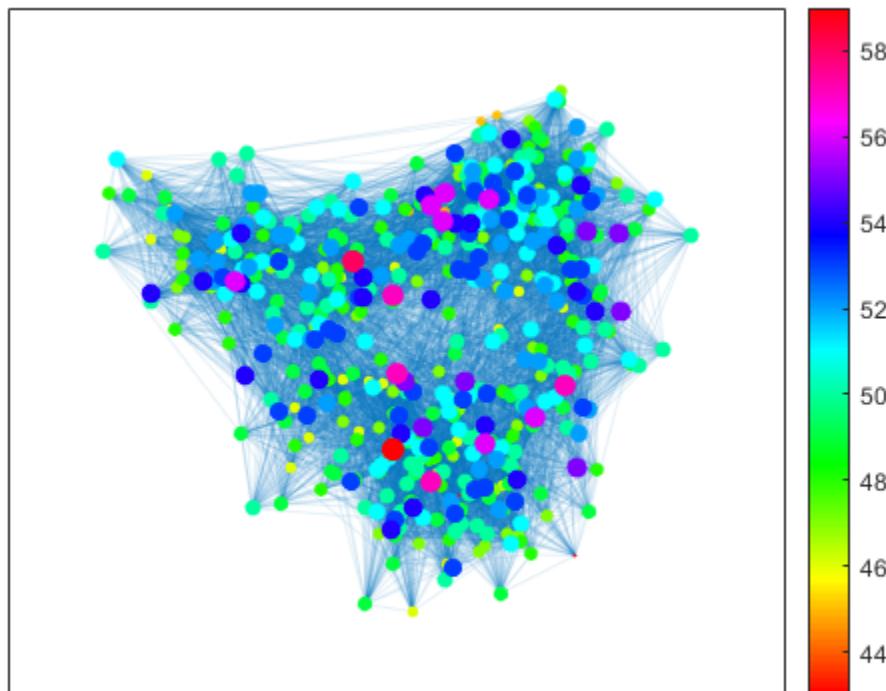
0.5	1.9101	85
1	1.9008	92

随着 **beta** 的增大，图形中的平均路径长度很快减少到其极限值。这是由于形成了紧密连接的枢纽节点，枢纽节点的数量将随着 **beta** 的增大而变得更多。

绘制 $\beta = 0.15$ Watts-Strogatz 模型图，使每个节点的大小和颜色与其出入度成比例。这是使枢纽形成过程可视化的有效方式。

```
colormap hsv
deg = degree(h2);
nSizes = 2*sqrt(deg-min(deg)+0.2);
nColors = deg;
plot(h2,'MarkerSize',nSizes,'NodeCData',nColors,'EdgeAlpha',0.1)
title('Watts-Strogatz Graph with $N = 500$ nodes, $K = 25$, and $\beta = 0.15$', ...
    'Interpreter','latex')
colorbar
```

Watts-Strogatz Graph with $N = 500$ nodes, $K = 25$, and $\beta = 0.15$



另请参阅

`digraph | graph`

使用 PageRank 算法对网站进行排名

以下示例演示如何使用 PageRank 算法对多个网站进行排名。虽然 PageRank 算法最初是为确定搜索引擎结果排名而设计的，但也可以更广泛地应用到许多不同类型的图形中的节点。PageRank 得分根据每个图形节点与其他节点的连接方式，对该节点的相对重要性给出意见。

理论上，PageRank 得分是指某人随机点击每个网站上的链接时会进入任何特定网页的极限概率。因此，得分较高的网页在网络中是高度关联并且可以发现的，随机的 Web 浏览者更可能访问该网页。

算法说明

在 PageRank 算法的每一步中，每个网页的得分都会根据以下公式更新：

$$r = (1-P)/n + P * (A' * (r/d) + s/n);$$

- r 是 PageRank 得分的向量。
- P 是标量阻尼因子（通常为 0.85），这是随机浏览者点击当前网页上的链接而不是在另一随机网页上继续点击的概率。
- A' 是图形的邻接矩阵的转置。
- d 是包含图形中每个节点的出度的向量。对于没有外向边的节点， d 设置为 1。
- n 是图形中节点的标量数量。
- s 是无链接的网页的 PageRank 得分的标量总和。

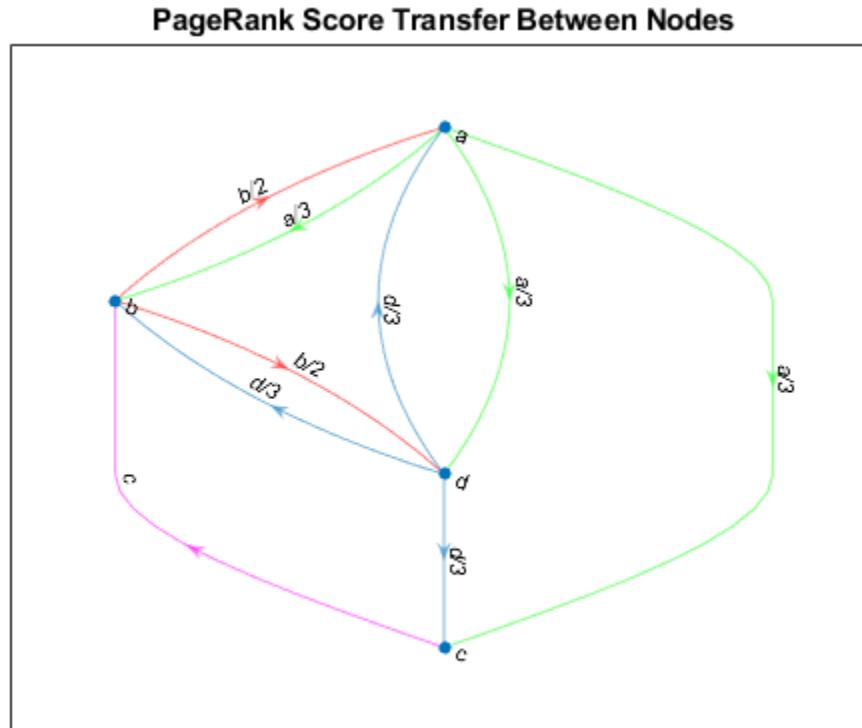
换言之，每个网页的排名很大程度上基于与之链接的网页的排名。项 $A' * (r/d)$ 会挑选出链接到图形中每个节点的源节点的得分，而得分按这些源节点的出站链接总数进行归一化。这确保了 PageRank 得分的总和始终为 1。例如，如果节点 2 链接到节点 1、3 和 4，则它会在算法的每次迭代期间向其中的每一个节点传送 1/3 的 PageRank 得分。

创建一个图形，用于说明每个节点如何将其 PageRank 得分赋予图形中的其他节点。

```

s = {'a' 'a' 'a' 'b' 'b' 'c' 'd' 'd' 'd'};
t = {'b' 'c' 'd' 'd' 'a' 'b' 'c' 'a' 'b'};
G = digraph(s,t);
labels = {'a/3' 'a/3' 'a/3' 'b/2' 'b/2' 'c' 'd/3' 'd/3' 'd/3'};
p = plot(G,'Layout','layered','EdgeLabel',labels);
highlight(p,[1 1],[2 3 4],'EdgeColor','g')
highlight(p,[2 2],[1 4],'EdgeColor','r')
highlight(p,3,2,'EdgeColor','m')
title('PageRank Score Transfer Between Nodes')

```



`centrality` 函数包含用于计算 PageRank 得分的选项。

包含 6 个节点的 PageRank

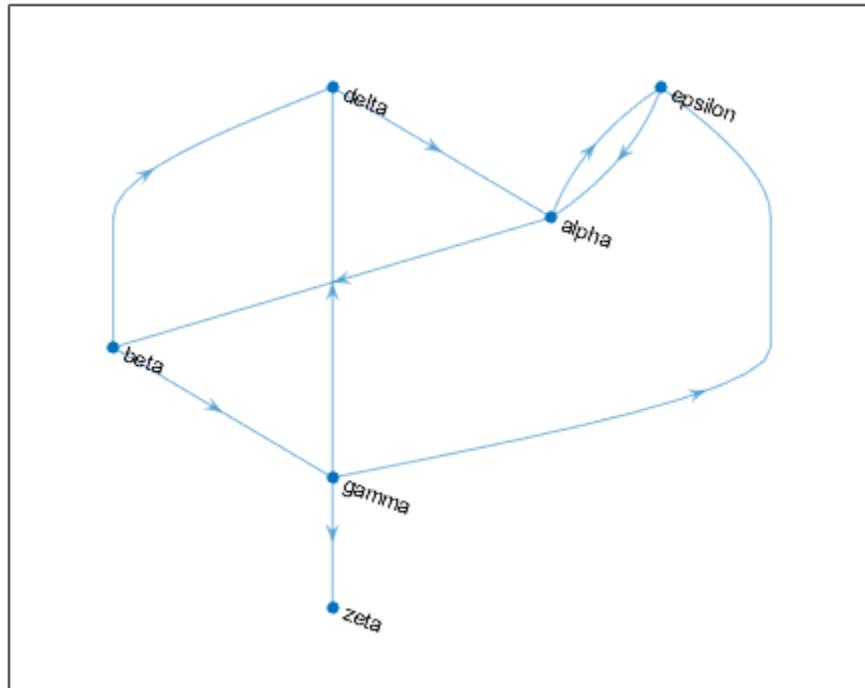
创建并绘制一个有向图，其中包含六个表示虚假网站的节点。

```
s = [1 1 2 2 3 3 3 4 5];
t = [2 5 3 4 4 5 6 1 1];
names = {'http://www.example.com/alpha', 'http://www.example.com/beta',...
    'http://www.example.com/gamma', 'http://www.example.com/delta',...
    'http://www.example.com/epsilon', 'http://www.example.com/zeta'};
G = digraph(s,t,[],names)

G =
digraph with properties:

Edges: [9x1 table]
Nodes: [6x1 table]

plot(G,'Layout','layered',...
'NodeLabel',{'alpha','beta','gamma','delta','epsilon','zeta'})
```



计算该图形的 PageRank 中心度得分。使用 0.85 的点进概率（或称为阻尼因子）。

```
pr = centrality(G,'pagerank','FollowProbability',0.85)
```

```
pr = 6×1
```

```
0.3210
0.1706
0.1066
0.1368
0.2008
0.0643
```

查看每个网页的 PageRank 得分和级别信息。

```
G.Nodes.PageRank = pr;
G.Nodes.InDegree = indegree(G);
G.Nodes.OutDegree = outdegree(G);
G.Nodes
```

```
ans=6×4 table
    Name        PageRank   InDegree   OutDegree
    _____
    {'http://www.example.com/alpha'}  0.32098      2          2
    {'http://www.example.com/beta'}   0.17057      1          2
    {'http://www.example.com/gamma'}  0.10657      1          3
```

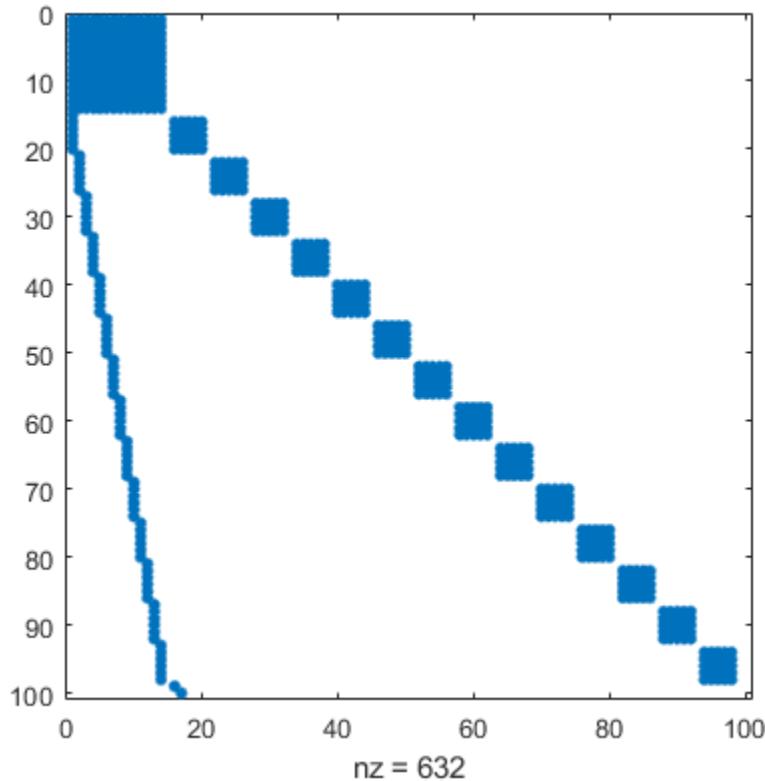
```
{'http://www.example.com/delta' } 0.13678    2      1
{'http://www.example.com/epsilon'} 0.20078    2      1
{'http://www.example.com/zeta' }   0.06432    1      0
```

结果显示，不仅仅是网页链接的数量可以决定得分，质量也是决定因素之一。**alpha** 和 **gamma** 网站的总级别都为 4，但 **alpha** 同时链接到 **epsilon** 和 **beta**，这两个网站也是排名较高的网站。**gamma** 仅链接到位于列表中间的一个页面 **beta**。因此，算法给出的 **alpha** 的得分高于 **gamma**。

mathworks.com 网站的 PageRank 得分

加载 **mathworks100.mat** 中的数据，并查看邻接矩阵 **A**。这些数据是使用自动网页爬网程序在 2015 年生成的。网页爬网程序从 <https://www.mathworks.com> 开始，然后是指向后续网页的链接，直到邻接矩阵包含 100 个唯一网页连接的相关信息。

```
load mathworks100.mat
spy(A)
```



通过将 **U** 中包含的 URL 用作节点名称，创建带有稀疏邻接矩阵 **A** 的有向图。

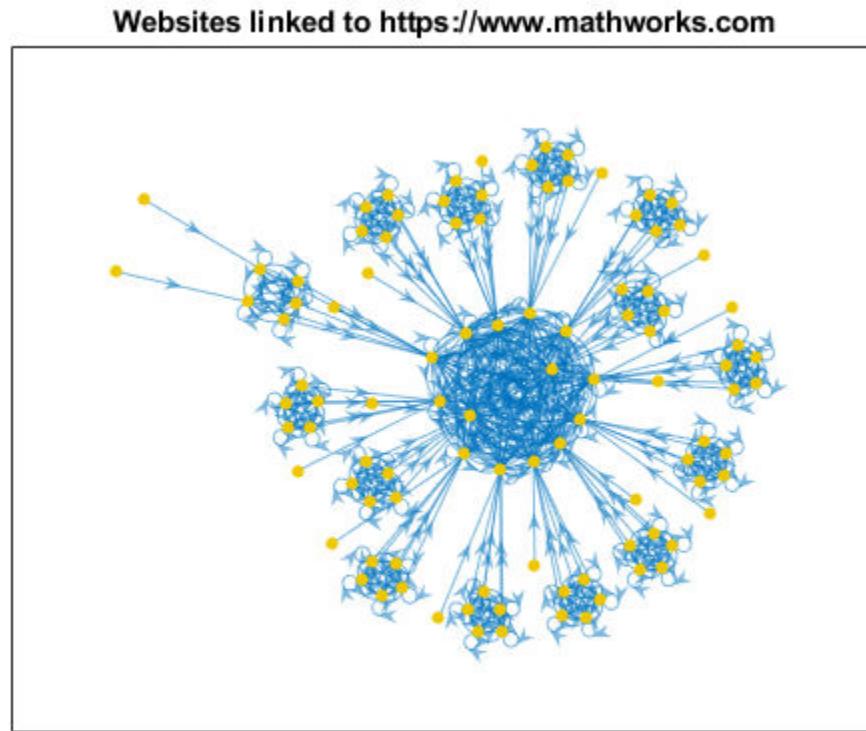
```
G = digraph(A,U)

G =
digraph with properties:

Edges: [632x1 table]
Nodes: [100x1 table]
```

使用强制布局绘制图表。

```
plot(G,'NodeLabel',[],'NodeColor',[0.93 0.78 0],'Layout','force');
title('Websites linked to https://www.mathworks.com')
```



使用 200 次迭代和阻尼因子 0.85 计算图形 G 的 PageRank 得分。将得分和级别信息添加到图形的节点表中。

```
pr = centrality(G,'pagerank','MaxIterations',200,'FollowProbability',0.85);
G.Nodes.PageRank = pr;
G.Nodes.InDegree = indegree(G);
G.Nodes.OutDegree = outdegree(G);
```

查看生成的前 25 个得分。

```
G.Nodes(1:25,:)
```

```
ans=25×4 table
```

Name	PageRank	InDegree	OutDegree
{'https://www.mathworks.com'}	0.044342	20	14
{'https://ch.mathworks.com'}	0.043085	20	14
{'https://cn.mathworks.com'}	0.043085	20	14
{'https://jp.mathworks.com'}	0.043085	20	14
{'https://kr.mathworks.com'}	0.043085	20	14
{'https://uk.mathworks.com'}	0.043085	20	14
{'https://au.mathworks.com'}	0.043085	20	14

```

{'https://de.mathworks.com'}          0.043085    20    14
{'https://es.mathworks.com'}           0.043085    20    14
{'https://fr.mathworks.com'}           0.043085    20    14
{'https://in.mathworks.com'}           0.043085    20    14
{'https://it.mathworks.com'}           0.043085    20    14
{'https://nl.mathworks.com'}           0.043085    20    14
{'https://se.mathworks.com'}           0.043085    20    14
{'https://www.mathworks.com/index.html%3Fnocookie%3Dtrue'} } 0.0015    0    1
{'https://www.mathworks.com/company/aboutus/policies_statements/patents.html'} 0.007714    6    6
:

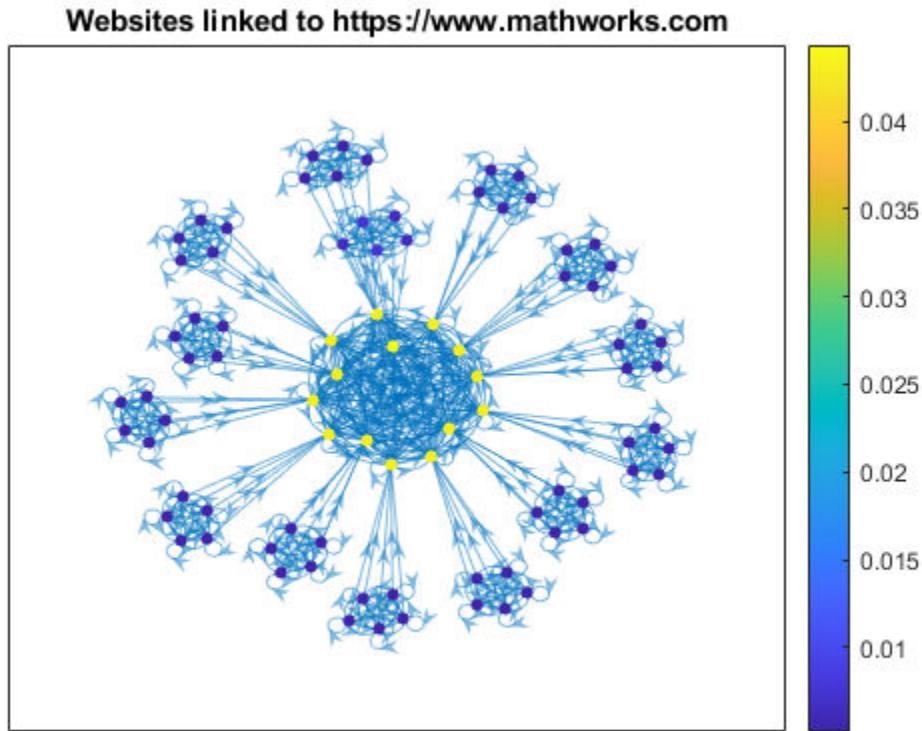
```

提取并绘制包含得分大于 0.005 的所有节点的子图。根据图形节点的 PageRank 得分为它们着色。

```

H = subgraph(G,find(G.Nodes.PageRank > 0.005));
plot(H,'NodeLabel',[],'NodeCData',H.Nodes.PageRank,'Layout','force');
title('Websites linked to https://www.mathworks.com')
colorbar

```



顶级网站的 PageRank 得分都十分相似，因此随机的 Web 浏览者大约有 4.5% 的几率登录每个网页。这一小组高度关联的网页在绘图中心形成了一个团。与这个中心团相连的是几个较小的团，这几个较小的团彼此之间高度关联。

参考

Moler, C. Experiments with MATLAB.Chapter 7:Google PageRank.MathWorks, Inc., 2011.

另请参阅

[centrality](#) | [digraph](#) | [graph](#)

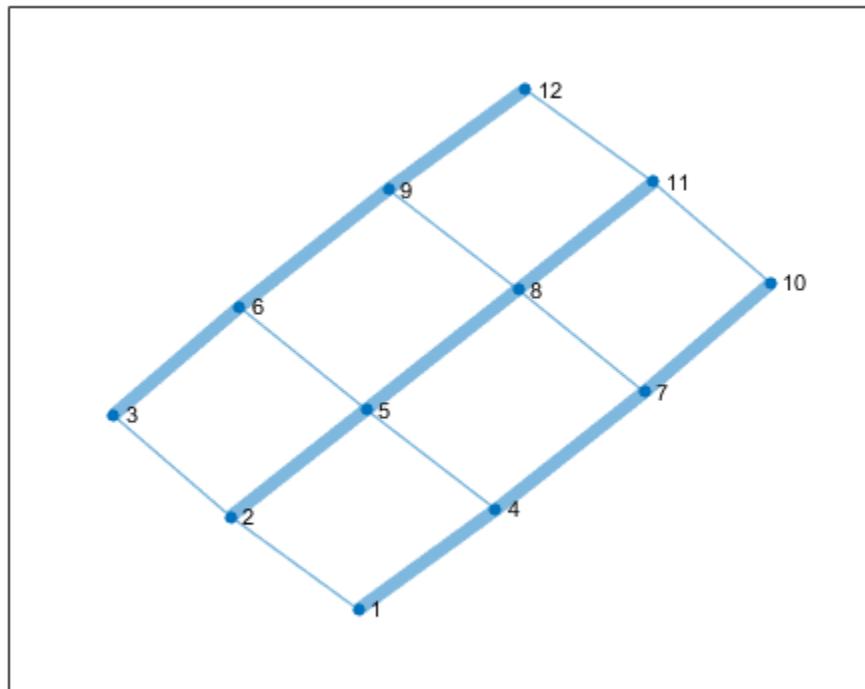
为图节点和边添加标签

此示例说明如何在图节点和图边上添加和自定义标签。

创建并绘制图

创建表示某个城市中网格化街道和交叉点的图。为边添加权重，使主干道和横穿街道在图中以不同的方式显示。使用与边权重成比例的边线宽绘图。

```
s = [1 1 2 2 3 4 4 5 5 6 7 7 8 8 9 10 11];
t = [2 4 3 5 6 5 7 6 8 9 8 10 9 11 12 11 12];
weights = [1 5 1 5 5 1 5 1 5 5 1 5 1 5 5 1 1];
G = graph(s,t,weights);
P = plot(G,'LineWidth',G.Edges.Weight);
```

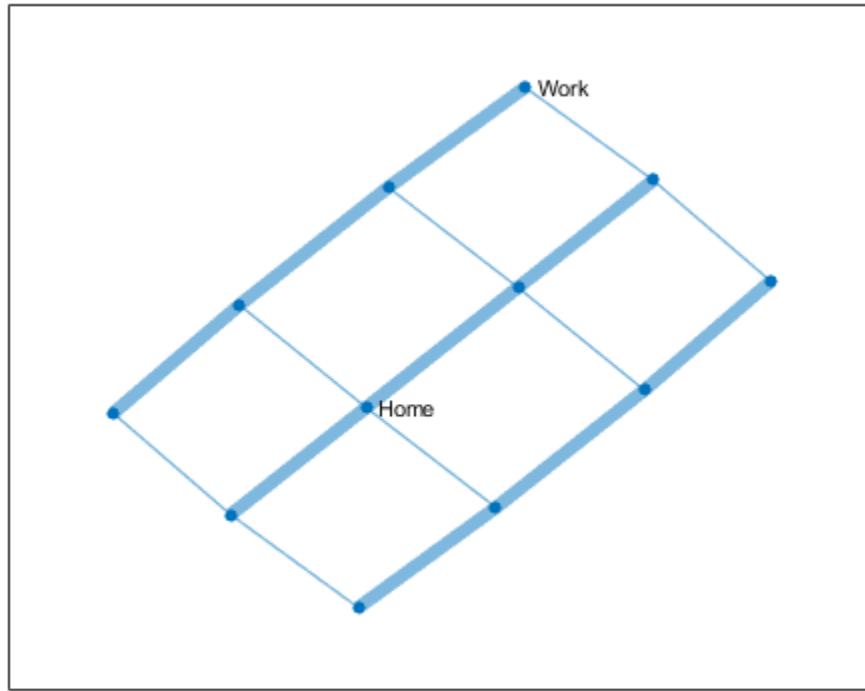


添加节点标签

对于节点数不超过 100 个的图，MATLAB® 会使用数字节点索引或节点名称自动标记节点（更大的图默认情况下将省略这些标签）。但是，您可以通过调整 `GraphPlot` 对象 `P` 的 `NodeLabel` 属性或使用 `labelnode` 函数来更改节点标签。因此，即使节点具有名称，也可以使用与这些名称不同的标签。

删除默认的数字节点标签。将一个交叉点标记为 `Home`，将另一个标记为 `Work`。

```
labelnode(P,1:12,"")
labelnode(P,5,'Home')
labelnode(P,12,'Work')
```



添加边标签

在绘制的图中，边不是自动标记的。您可以通过更改 `GraphPlot` 对象 `P` 的 `EdgeLabel` 属性值或使用 `labeledge` 函数来添加边标签。

为纽约市的街道添加边标签。边的顺序在图的 `G.Edges` 表中定义，因此您指定的标签顺序必须遵循该顺序。将边标签直接存储在 `G.Edges` 表中很方便，这样边名称就位于其他边信息的旁边。

G.Edges

```
ans=17×2 table
    EndNodes    Weight
    _____
    1     2      1
    1     4      5
    2     3      1
    2     5      5
    3     6      5
    4     5      1
    4     7      5
    5     6      1
    5     8      5
    6     9      5
    7     8      1
    7    10      5
    8     9      1
    8    11      5
```

```

9   12   5
10  11   1
:

```

下面的示例有 17 条边，但只有 7 个唯一的街道名称。因此，可以在元胞数组中定义街道名称，然后对元胞数组进行索引，以检索每条边需要的街道名称。将变量添加到包含街道名称的 **G.Edges** 表中。

```

streets = {'8th Ave' '7th Ave' '6th Ave' '5th Ave' ...
    'W 20th St' 'W 21st St' 'W 22nd St'};
inds = [1 5 1 6 7 2 5 2 6 7 3 5 3 6 7 4 4];
G.Edges.StreetName = streets(inds);
G.Edges

```

```

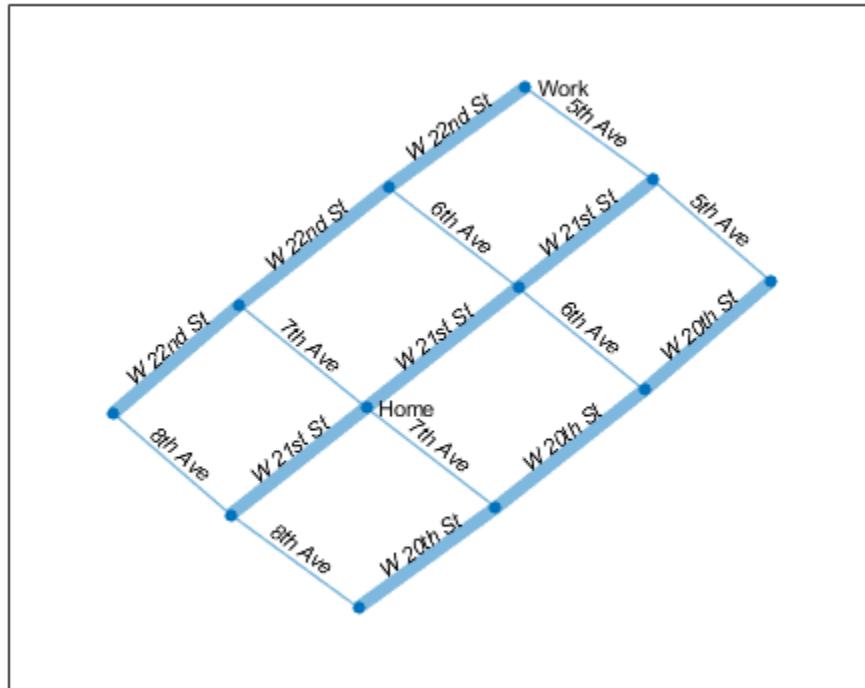
ans=17×3 table
EndNodes  Weight  StreetName

```

1	2	1	{'8th Ave' }
1	4	5	{'W 20th St'}
2	3	1	{'8th Ave' }
2	5	5	{'W 21st St'}
3	6	5	{'W 22nd St'}
4	5	1	{'7th Ave' }
4	7	5	{'W 20th St'}
5	6	1	{'7th Ave' }
5	8	5	{'W 21st St'}
6	9	5	{'W 22nd St'}
7	8	1	{'6th Ave' }
7	10	5	{'W 20th St'}
8	9	1	{'6th Ave' }
8	11	5	{'W 21st St'}
9	12	5	{'W 22nd St'}
10	11	1	{'5th Ave' }
:			

更新 **EdgeLabel** 属性，以引用这些街道名称。

```
P.EdgeLabel = G.Edges.StreetName;
```



调整字体属性

图论图中的节点标签和边标签具有各自的属性，它们控制着标签的外观和样式。由于属性是分离的，因此可以对节点标签和边标签使用不同的样式。

对于**节点**标签，可以调整：

- **NodeLabel**
- **NodeLabelColor**
- **NodeFontName**
- **NodeFontSize**
- **NodeFontWeight**
- **NodeFontAngle**

对于**边**标签，可以调整：

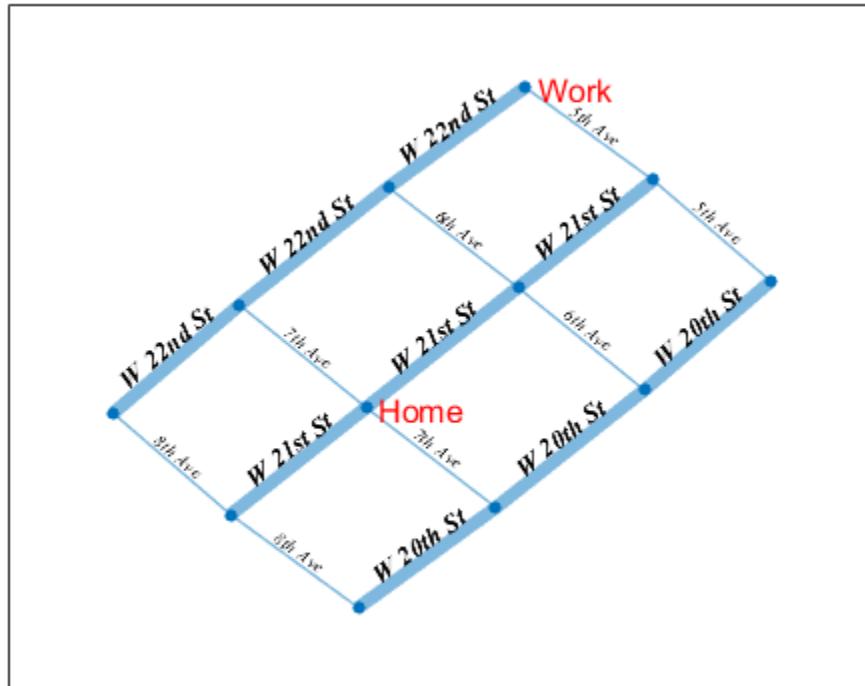
- **EdgeLabel**
- **EdgeLabelColor**
- **EdgeFontName**
- **EdgeFontSize**
- **EdgeFontWeight**
- **EdgeFontAngle**

使用这些属性，可以调整此示例中纽约市街道使用的字体：

- 更改 `NodeFontSize` 和 `NodeLabelColor`，使交叉点标签的字体为 12 磅，颜色为红色。
- 更改 `EdgeFontWeight`、`EdgeFontAngle` 和 `EdgeFontSize`，为一个方向的街道使用较大的粗体字体，为另一个方向的街道使用较小的斜体字体。
- 更改 `EdgeFontName`，使用 Times New Roman 作为边标签。

可以使用 `highlight` 函数更改图边子集的图属性。创建逻辑索引 `isAvenue`，对于包含单词 'Ave' 的边标签，逻辑索引的值为 `true`。使用此逻辑向量作为 `highlight` 的输入，以一种方式标记所有主干道，以另一种方式标记所有非主干道。

```
P.NodeFontSize = 12;
P.NodeLabelColor = 'r';
isAvenue = contains(P.EdgeLabel, 'Ave');
highlight(P, 'Edges', isAvenue, 'EdgeFontAngle', 'italic', 'EdgeFontSize', 7);
highlight(P, 'Edges', ~isAvenue, 'EdgeFontWeight', 'bold', 'EdgeFontSize', 10);
P.EdgeFontName = 'Times New Roman';
```



突出显示边

找到 Home 和 Work 节点之间的最短路线，并检查哪些街道在该路线上。以红色突出显示该路线上的节点和边，并删除不在该路线上的所有边的边标签。

```
[path,d,pathEdges] = shortestpath(G,5,12)
```

```
path = 1×4
```

5 6 9 12

$d = 11$

`pathEdges = 1×3`

8 10 15

`G.Edges.StreetName(pathEdges,:)`

`ans = 3x1 cell`

`{'7th Ave' }`

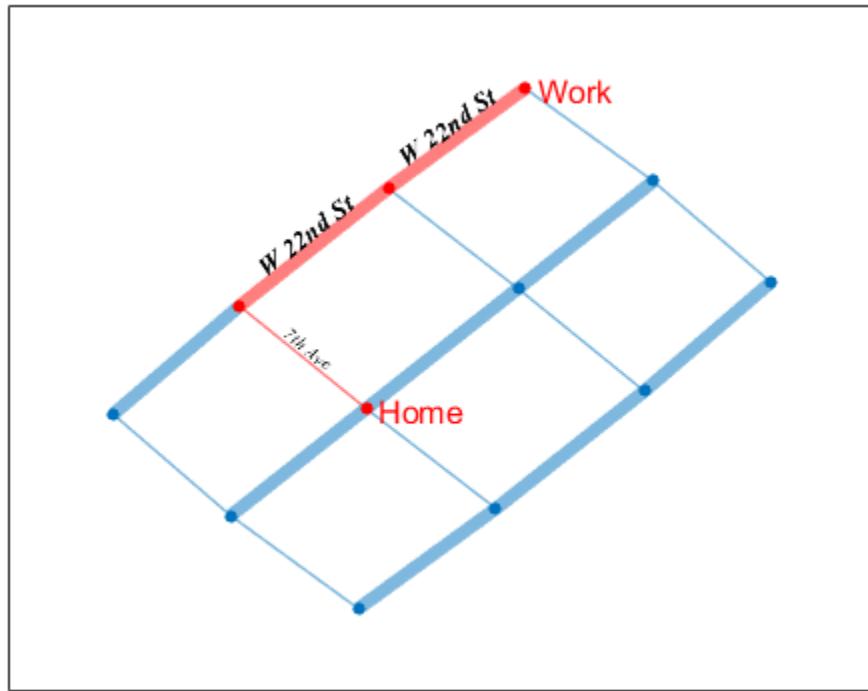
`{'W 22nd St'}`

`{'W 22nd St'}`

`highlight(P,'Edges',pathEdges,'EdgeColor','r')`

`highlight(P,path,'NodeColor','r')`

`labeledge(P, setdiff(1:numedges(G), pathEdges), "r")`



另请参阅

`GraphPlot`

详细信息

- “图的绘制和自定义” (第 5-17 页)
- “将节点属性添加到图论图数据游标” (第 5-36 页)

一元函数

- “创建并计算多项式” (第 6-2 页)
- “多项式的根” (第 6-4 页)
- “对多项式求积分和微分” (第 6-9 页)
- “多项式曲线拟合” (第 6-11 页)
- “预测美国人口” (第 6-13 页)
- “标量函数的根” (第 6-18 页)

创建并计算多项式

此示例说明如何在 MATLAB® 中将多项式表示为向量以及根据相关点计算多项式。

表示多项式

MATLAB® 将多项式表示为行向量，其中包含按降幂排序的系数。例如，三元素向量

```
p = [p2 p1 p0];
```

表示多项式

$$p(x) = p_2x^2 + p_1x + p_0.$$

创建一个向量以表示二次多项式 $p(x) = x^2 - 4x + 4$ 。

```
p = [1 -4 4];
```

此外，还必须将系数为 0 的多项式中间项输入到该向量中，因为 0 用作 x 的特定幂的占位符。

创建一个向量来表示多项式 $p(x) = 4x^5 - 3x^2 + 2x + 33$ 。

```
p = [4 0 0 -3 2 33];
```

多项式的计算

将多项式作为向量输入到 MATLAB® 后，请使用 **polyval** 函数根据特定值计算多项式。

使用 **polyval** 计算 $p(2)$ 。

```
polyval(p,2)
```

```
ans = 153
```

您也可以使用 **polyvalm** 以矩阵方式计算多项式。一个变量中的多项式表达式 $p(x) = 4x^5 - 3x^2 + 2x + 33$ 将变为矩阵表达式

$$p(X) = 4X^5 - 3X^2 + 2X + 33I,$$

其中，X 是方阵，I 是单位矩阵。

创建方阵 X 并根据 X 计算 p。

```
X = [2 4 5; -1 0 3; 7 1 5];
Y = polyvalm(p,X)
```

```
Y = 3×3
```

154392	78561	193065
49001	24104	59692
215378	111419	269614

另请参阅

[poly](#) | [polyval](#) | [polyvalm](#) | [roots](#)

详细信息

- “多项式的根” (第 6-4 页)
- “对多项式求积分和微分” (第 6-9 页)
- “多项式曲线拟合” (第 6-11 页)

多项式的根

此示例演示如何通过多种不同的方法计算多项式的根。

本节内容

- “数值根” (第 6-4 页)
- “使用代换法求根” (第 6-4 页)
- “特定区间内的根” (第 6-5 页)
- “符号根” (第 6-7 页)

数值根

roots 函数用于计算系数向量表示的单变量多项式的根。

例如，创建一个向量以表示多项式 $x^2 - x - 6$ ，然后计算多项式的根。

```
p = [1 -1 -6];
r = roots(p)
```

r =

```
3
-2
```

按照惯例，MATLAB 以列向量形式返回这些根。

poly 函数将这些根重新转换为多项式系数。对向量执行运算时，**poly** 和 **roots** 为反函数，因此 **poly(roots(p))** 返回 p (取决于舍入误差、排序和缩放)。

```
p2 = poly(r)
```

p2 =

```
1 -1 -6
```

对矩阵执行运算时，**poly** 函数会计算矩阵的特征多项式。特征多项式的根是矩阵的特征值。因此，**roots(poly(A))** 和 **eig(A)** 返回相同的答案 (取决于舍入误差、排序和缩放)。

使用代换法求根

您可以通过使用代换法简化方程来对涉及三角函数的多项式方程求解。一个变量的生成多项式不再包含任何三角函数。

例如，计算 θ 用于对该方程进行求解的值

$$3\cos^2(\theta) - \sin(\theta) + 3 = 0.$$

利用 $\cos^2(\theta) = 1 - \sin^2(\theta)$ ，完全以正弦函数的方式表示该方程：

$$-3\sin^2(\theta) - \sin(\theta) + 6 = 0.$$

利用代换法 $x = \sin(\theta)$ ，将该方程表示为简单的多项式方程：

$$-3x^2 - x + 6 = 0.$$

创建一个向量以表示多项式。

```
p = [-3 -1 6];
```

求多项式的根。

```
r = roots(p)
```

```
r = 2×1
```

```
-1.5907  
1.2573
```

要撤消代换法，请使用 $\theta = \sin^{-1}(x)$ 。**asin** 函数计算反正弦。

```
theta = asin(r)
```

```
theta = 2×1 complex
```

```
-1.5708 + 1.0395i  
1.5708 - 0.7028i
```

验证 **theta** 中的元素是否为 θ 中用来对原始方程求解的值（在舍入误差内）。

```
f = @(Z) 3*cos(Z).^2 - sin(Z) + 3;  
f(theta)
```

```
ans = 2×1 complex  
10-14 ×
```

```
-0.0888 + 0.0647i  
0.2665 + 0.0399i
```

特定区间内的根

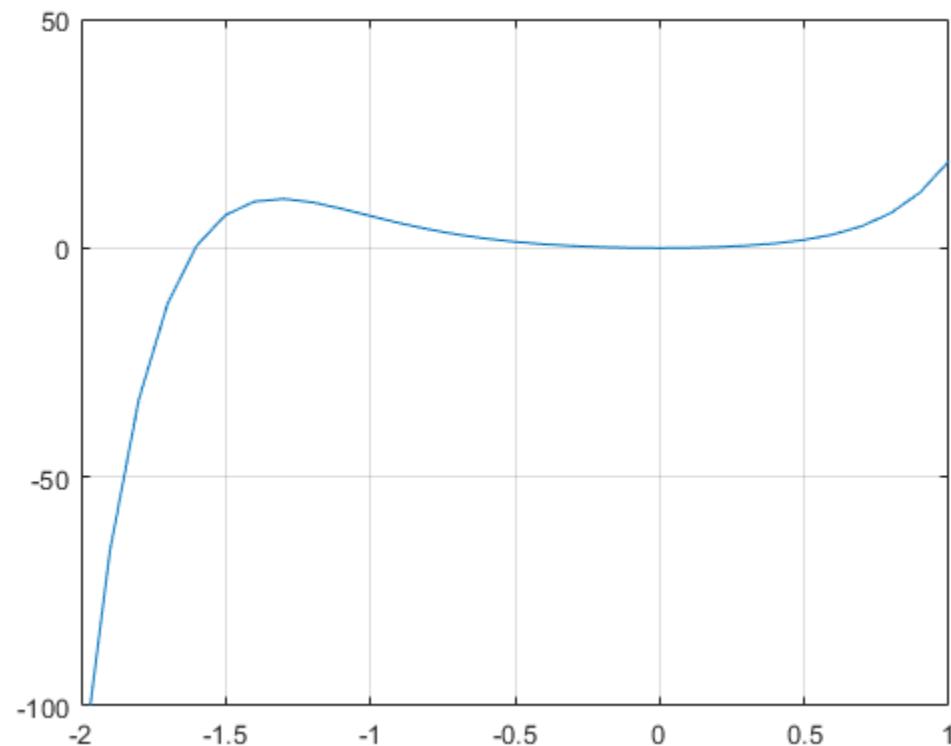
使用 **fzero** 函数求多项式在特定区间内的根。在其他使用情况下，如果您绘制多项式并想要知道特定根的值，则这种方法很适用。

例如，创建一个函数句柄以表示多项式 $3x^7 + 4x^6 + 2x^5 + 4x^4 + x^3 + 5x^2$ 。

```
p = @(x) 3*x.^7 + 4*x.^6 + 2*x.^5 + 4*x.^4 + x.^3 + 5*x.^2;
```

在区间 $[-2, 1]$ 内绘制该函数。

```
x = -2:0.1:1;  
plot(x,p(x))  
ylim([-100 50])  
grid on  
hold on
```

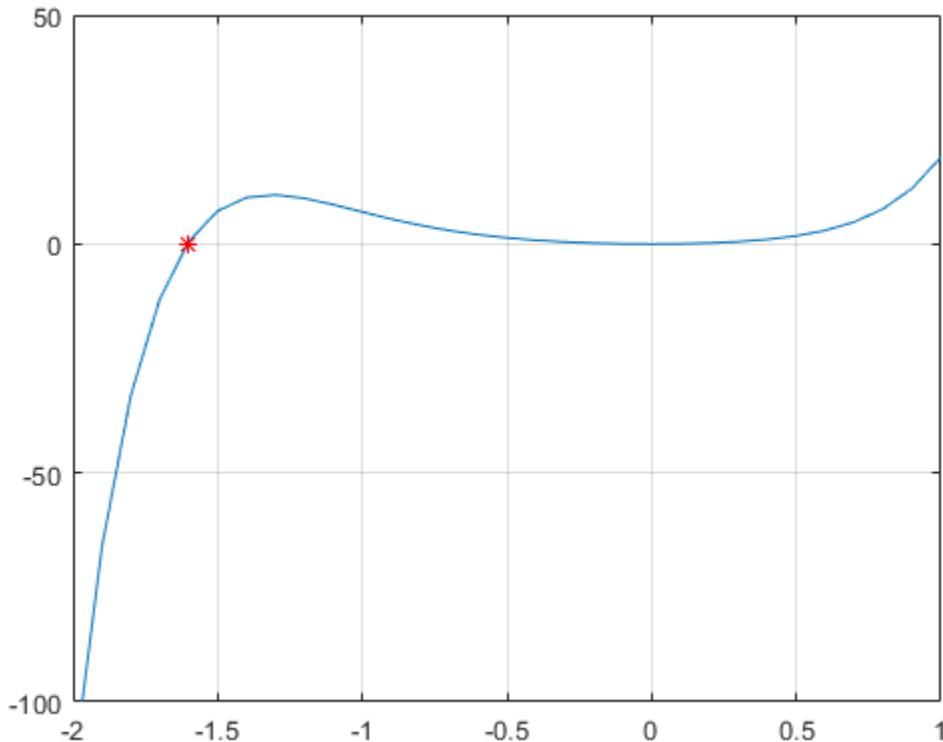


从绘图中，多项式在 0 和另一个接近 -1.5 的位置各有一个简单的根。使用 `fzero` 计算并绘制接近 -1.5 的根。

```
Z = fzero(p, -1.5)
```

```
Z = -1.6056
```

```
plot(Z,p(Z),'r*')
```



符号根

如果您有 Symbolic Math Toolbox™，则还会提供以符号形式计算多项式的其他选项。一种方式是使用 **solve** 函数。

```
syms x
s = solve(x^2-x-6)

s =
-2
3
```

另一种方式是使用 **factor** 函数计算多项式各项的因子。

```
F = factor(x^2-x-6)

F =
[x + 2, x - 3]
```

有关详细信息，请参阅 “Solve Algebraic Equation” (Symbolic Math Toolbox)。

另请参阅

eig | **poly** | **roots**

详细信息

- “创建并计算多项式” (第 6-2 页)
- “标量函数的根” (第 6-18 页)
- “对多项式求积分和微分” (第 6-9 页)

对多项式求积分和微分

此示例演示如何使用 **polyint** 和 **polyder** 函数对由系数向量表示的任何多项式求解析积分或微分。

使用 **polyder** 获取多项式 $p(x) = x^3 - 2x - 5$ 的导数。生成的多项式为 $q(x) = \frac{d}{dx}p(x) = 3x^2 - 2$ 。

```
p = [1 0 -2 -5];
q = polyder(p)
```

$q = 1 \times 3$

3 0 -2

同样，使用 **polyint** 对多项式 $p(x) = 4x^3 - 3x^2 + 1$ 求积分。生成的多项式为

$$q(x) = \int p(x)dx = x^4 - x^3 + x.$$

```
p = [4 -3 0 1];
q = polyint(p)
```

$q = 1 \times 5$

1 -1 0 1 0

polyder 也可以计算两个多项式积或商的导数。例如，创建两个向量来表示多项式 $a(x) = x^2 + 3x + 5$ 和 $b(x) = 2x^2 + 4x + 6$ 。

```
a = [1 3 5];
b = [2 4 6];
```

通过调用带有单个输出参数的 **polyder** 来计算导数 $\frac{d}{dx}[a(x)b(x)]$ 。

c = polyder(a,b)

$c = 1 \times 4$

8 30 56 38

通过调用带有两个输出参数的 **polyder** 来计算导数 $\frac{d}{dx}\left[\frac{a(x)}{b(x)}\right]$ 。生成的多项式为

$$\frac{d}{dx}\left[\frac{a(x)}{b(x)}\right] = \frac{-2x^2 - 8x - 2}{4x^4 + 16x^3 + 40x^2 + 48x + 36} = \frac{q(x)}{d(x)}.$$

[q,d] = polyder(a,b)

$q = 1 \times 3$

-2 -8 -2

$d = 1 \times 5$

4 16 40 48 36

另请参阅

[conv](#) | [deconv](#) | [polyder](#) | [polyint](#)

详细信息

- “多项式积分的解析解” (第 15-7 页)
- “创建并计算多项式” (第 6-2 页)

多项式曲线拟合

此示例说明如何使用 **polyfit** 函数将多项式曲线与一组数据点拟合。您可以按照以下语法，使用 **polyfit** 求出以最小二乘方式与一组数据拟合的多项式的系数

```
p = polyfit(x,y,n);
```

其中：

- **x** 和 **y** 是包含数据点的 **x** 和 **y** 坐标的向量
- **n** 是要拟合的多项式的次数

创建包含五个数据点的 **x-y** 测试数据。

```
x = [1 2 3 4 5];
y = [5.5 43.1 128 290.7 498.4];
```

使用 **polyfit** 求与数据近似拟合的三次多项式。

```
p = polyfit(x,y,3)
```

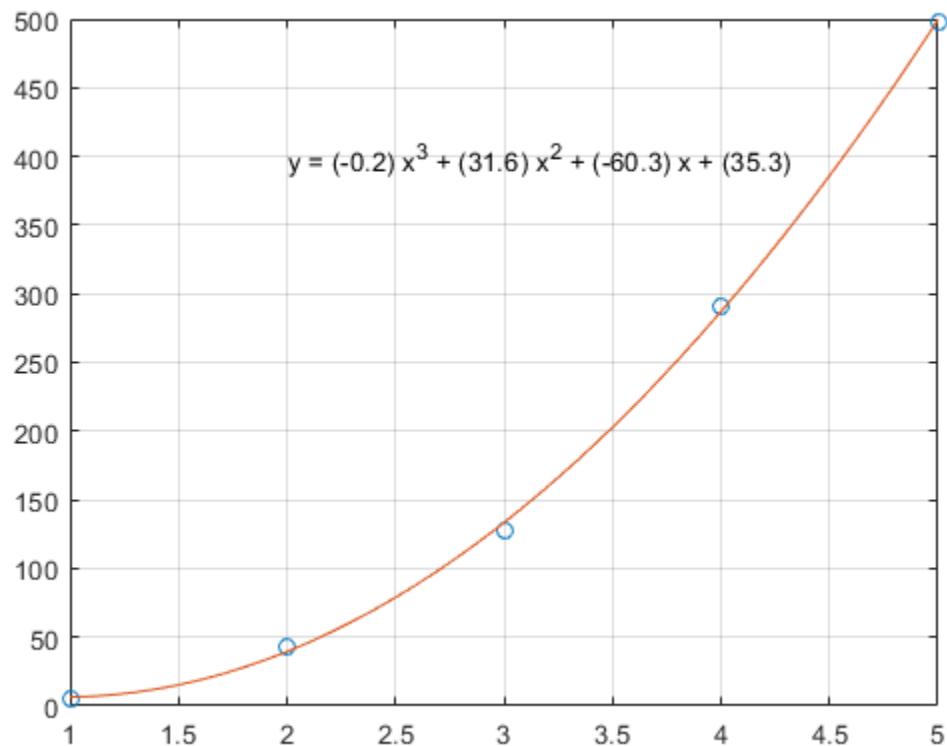
```
p = 1×4
```

```
-0.1917 31.5821 -60.3262 35.3400
```

使用 **polyfit** 获取拟合线的多项式后，可以使用 **polyval** 计算可能未包含在原始数据中的其他点处的多项式。

在更小域内计算 **polyfit** 估计值，并绘制实际数据值的估计值以进行对比。可以为拟合线包含方程注释。

```
x2 = 1:.1:5;
y2 = polyval(p,x2);
plot(x,y,'o',x2,y2)
grid on
s = sprintf('y = (%.1f) x^3 + (%.1f) x^2 + (%.1f) x + (%.1f)',p(1),p(2),p(3),p(4));
text(2,400,s)
```



另请参阅

[polyfit](#) | [polyval](#)

详细信息

- “以编程方式拟合”
- “创建并计算多项式” (第 6-2 页)

预测美国人口

此示例说明，即使使用次数最适中的多项式外插数据也是有风险且不可靠的。

此示例比 MATLAB® 出现得更早。该示例最初作为一个练习出现在 Forsythe、Malcolm 和 Moler 合著的《Computer Methods for Mathematical Computations》一书中，该书由出版商 Prentice-Hall 在 1977 年出版。

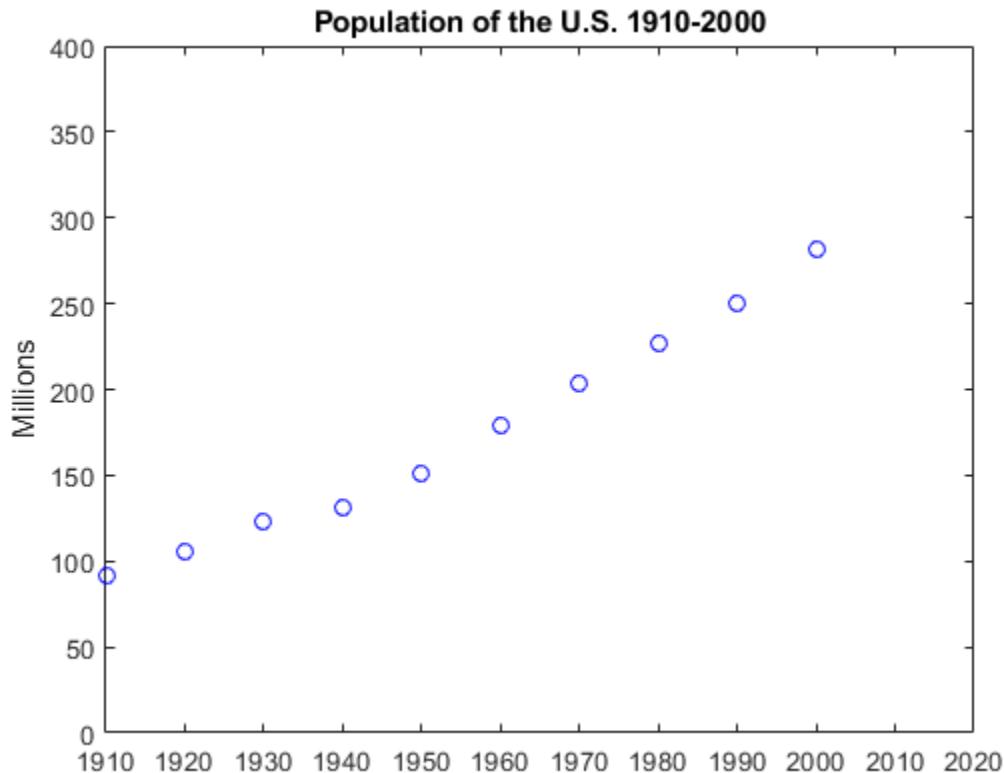
现在，通过 MATLAB 可以更容易地改变参数和查看结果，但基础数学原理未变。

使用 1910 年至 2000 年的美国人口普查数据创建并绘制两个向量。

```
% Time interval
t = (1910:10:2000)';

% Population
p = [91.972 105.711 123.203 131.669 150.697...
    179.323 203.212 226.505 249.633 281.422]';

% Plot
plot(t,p,'bo');
axis([1910 2020 0 400]);
title('Population of the U.S. 1910-2000');
ylabel('Millions');
```



那么猜想一下 2010 年美国的人口是多少？

```
p
p = 10×1
```

```
91.9720
105.7110
123.2030
131.6690
150.6970
179.3230
203.2120
226.5050
249.6330
281.4220
```

将这些数据与 t 中的一个多项式拟合，并使用它将人口数外插到 $t = 2010$ 。通过对包含 Vandermonde 矩阵的线性方程组求解来获得多项式中的系数，该矩阵为 11×11 ，其元素为缩放时间的幂，即 $A(i,j) = s(i)^{(n-j)}$ 。

```
n = length(t);
s = (t-1950)/50;
A = zeros(n);
A(:,end) = 1;
for j = n-1:-1:1
    A(:,j) = s .* A(:,j+1);
end
```

通过对包含 Vandermonde 矩阵最后 $d+1$ 列的线性方程组求解，获得与数据 p 拟合的 d 次多项式的系数 c ：

```
A(:,n-d:n)*c ~= p
```

- 如果 $d < 10$ ，则方程个数多于未知数个数，并且最小二乘解是合适的。
- 如果 $d == 10$ ，则可以精确求解方程，而多项式实际上会对数据进行插值。

在任一种情况下，都可以使用反斜杠运算符来求解方程组。三次拟合的系数为：

```
c = A(:,n-3:n)\p
```

```
c = 4×1
```

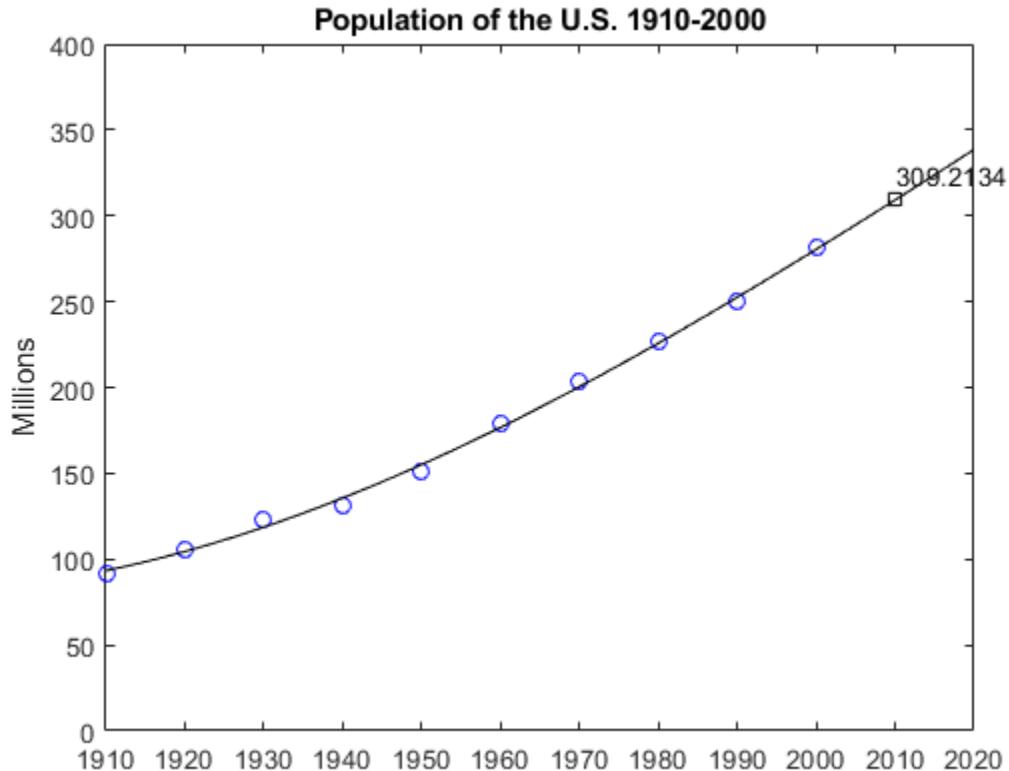
```
-5.7042
27.9064
103.1528
155.1017
```

现在，计算从 1910 年到 2010 年每一年的多项式，然后绘制结果。

```
v = (1910:2020)';
x = (v-1950)/50;
w = (2010-1950)/50;
y = polyval(c,x);
z = polyval(c,w);

hold on
plot(v,y,'k-');
```

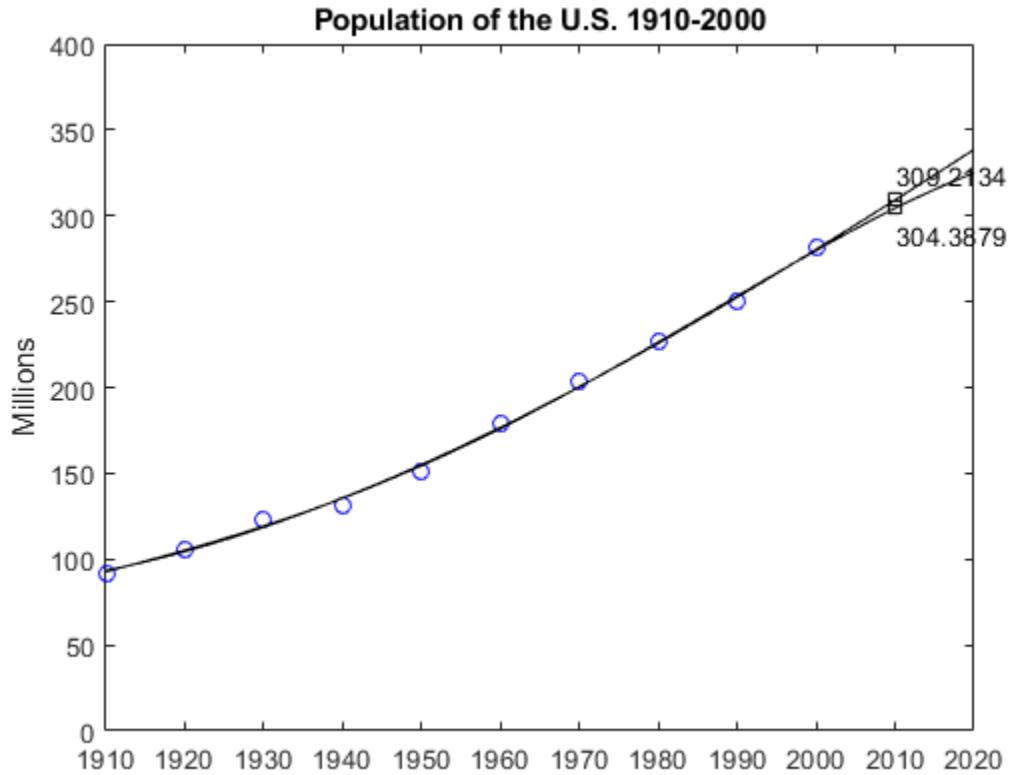
```
plot(2010,z,'ks');
text(2010,z+15,num2str(z));
hold off
```



将三次拟合与四次拟合进行比较。请注意，外插点完全不同。

```
c = A(:,n-4:n)\p;
y = polyval(c,x);
z = polyval(c,w);

hold on
plot(v,y,'k-');
plot(2010,z,'ks');
text(2010,z-15,num2str(z));
hold off
```

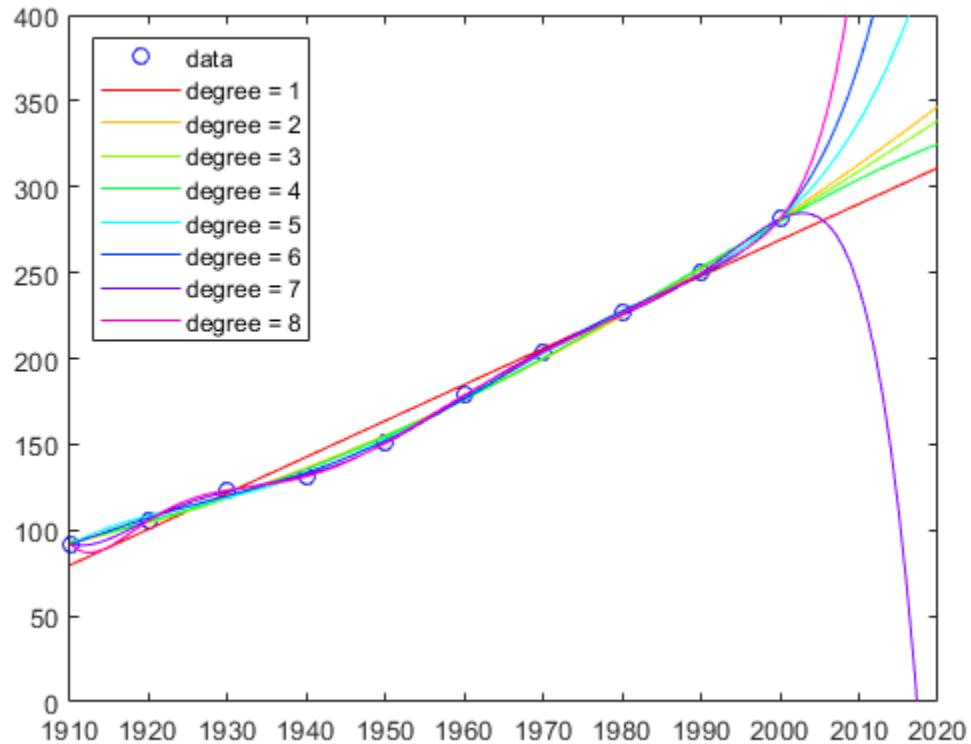


随着阶数增加，外插变得越来越不可靠。

```

cla
plot(t,p,'bo')
hold on
axis([1910 2020 0 400])
colors = hsv(8);
labels = {'data'};
for d = 1:8
    [Q,R] = qr(A(:,n-d:n));
    R = R(1:d+1,:);
    Q = Q(:,1:d+1);
    c = R\Q'*p; % Same as c = A(:,n-d:n)\p;
    y = polyval(c,x);
    z = polyval(c,11);
    plot(v,y,'color',colors(d,:));
    labels{end+1} = ['degree = ' int2str(d)];
end
legend(labels, 'Location', 'NorthWest')
hold off

```



另请参阅

polyfit

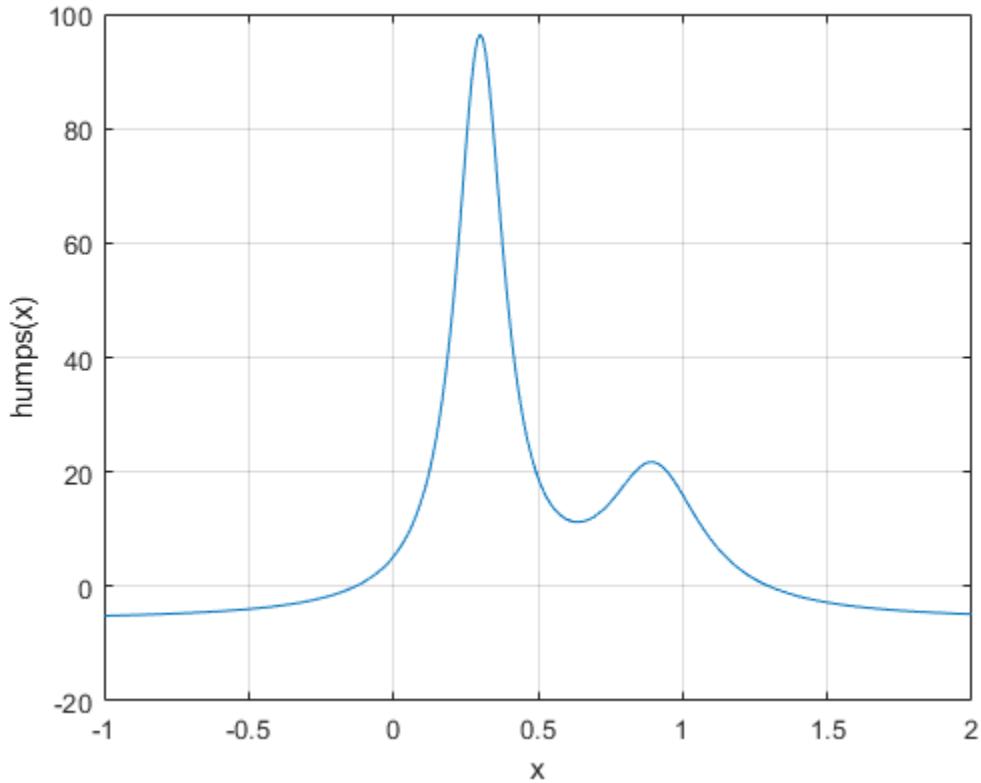
标量函数的根

对一元非线性方程求解

fzero 函数尝试求一个一元方程的根。可以通过用于指定起始区间的单元素起点或双元素向量调用该函数。如果为 **fzero** 提供起点 **x0**, **fzero** 将首先搜索函数更改符号的点周围的区间。如果找到该区间, **fzero** 返回函数更改符号的位置附近的值。如果未找到此类区间, **fzero** 返回 **NaN**。或者, 如果知道函数值的符号不同的两个点, 可以使用双元素向量指定该起始区间; **fzero** 保证缩小该区间并返回符号更改处附近的值。

以下部分包含两个示例, 用于说明如何使用起始区间和起点查找函数的零元素。这些示例使用由 MATLAB® 提供的函数 **humps.m**。下图显示了 **humps** 的图。

```
x = -1:0.01:2;
y = humps(x);
plot(x,y)
xlabel('x')
ylabel('humps(x)')
grid on
```



为 **fzero** 设置选项

可以通过设置选项控制 **fzero** 函数的多个方面。使用 **optimset** 设置选项。选项包括:

- 选择 **fzero** 生成的显示量 - 请参阅“设置优化选项” (第 9-8 页)、使用起始区间 (第 6-0 页) 和使用起点 (第 6-0 页)。

- 选择控制 **fzero** 如何确定它得到根的不同公差 - 请参阅“设置优化选项”（第 9-8 页）。
- 选择用于观察 **fzero** 逼近根的进度的绘图函数 - 请参阅“优化求解器绘制函数”（第 9-17 页）。
- 使用自定义编程的输出函数观察 **fzero** 逼近根的进度 - 请参阅“优化求解器输出函数”（第 9-12 页）。

使用起始区间

humps 的图指示 $x = -1$ 时函数为负数， $x = 1$ 时函数为正数。可以通过计算这两点的 **humps** 进行确认。

```
humps(1)
```

```
ans = 16
```

```
humps(-1)
```

```
ans = -5.1378
```

因此，可以将 $[-1 1]$ 用作 **fzero** 的起始区间。

fzero 的迭代算法可求 $[-1 1]$ 越来越小的子区间。对于每个子区间，**humps** 在两个端点的符号不同。由于子区间的端点彼此越来越近，因此它们收敛到 **humps** 的零位置。

要显示 **fzero** 在每个迭代过程中的进度，请使用 **optimset** 函数将 **Display** 选项设置为 **iter**。

```
options = optimset('Display','iter');
```

然后如下所示调用 **fzero**：

```
a = fzero(@humps,[-1 1],options)
```

Func-count	x	f(x)	Procedure
2	-1	-5.13779	initial
3	-0.513876	-4.02235	interpolation
4	-0.513876	-4.02235	bisection
5	-0.473635	-3.83767	interpolation
6	-0.115287	0.414441	bisection
7	-0.115287	0.414441	interpolation
8	-0.132562	-0.0226907	interpolation
9	-0.131666	-0.0011492	interpolation
10	-0.131618	1.88371e-07	interpolation
11	-0.131618	-2.7935e-11	interpolation
12	-0.131618	8.88178e-16	interpolation
13	-0.131618	8.88178e-16	interpolation

Zero found in the interval [-1, 1]

```
a = -0.1316
```

每个值 **x** 代表迄今为止最佳的端点。**Procedure** 列向您显示每步的算法是使用对分还是插值。

可以通过输入以下内容验证 **a** 中的函数值是否接近零：

```
humps(a)
```

```
ans = 8.8818e-16
```

起点的使用

假定您不知道 **humps** 的函数值符号不同的两点。在这种情况下，可以选择标量 **x0** 作为 **fzero** 的起点。**fzero** 先搜索函数更改符号的点附近的区间。如果 **fzero** 找到此类区间，它会继续执行上一部分中介绍的算法。如果未找到此类区间，**fzero** 返回 **NaN**。

例如，将起点设置为 **-0.2**，将 **Display** 选项设置为 **Iter**，并调用 **fzero**：

```
options = optimset('Display','iter');
a = fzero(@humps,-0.2,options)
```

```
Search for an interval around -0.2 containing a sign change:
Func-count    a      f(a)      b      f(b)  Procedure
    1        -0.2    -1.35385    -0.2    -1.35385 initial interval
    3     -0.194343    -1.26077    -0.205657    -1.44411 search
    5      -0.192    -1.22137    -0.208    -1.4807 search
    7     -0.188686    -1.16477    -0.211314    -1.53167 search
    9      -0.184    -1.08293    -0.216    -1.60224 search
   11     -0.177373    -0.963455    -0.222627    -1.69911 search
   13      -0.168    -0.786636    -0.232    -1.83055 search
   15     -0.154745    -0.51962    -0.245255    -2.00602 search
   17      -0.136    -0.104165    -0.264    -2.23521 search
   18     -0.10949     0.572246    -0.264    -2.23521 search
```

Search for a zero in the interval [-0.10949, -0.264]:

Func-count	x	f(x)	Procedure
18	-0.10949	0.572246	initial
19	-0.140984	-0.219277	interpolation
20	-0.132259	-0.0154224	interpolation
21	-0.131617	3.40729e-05	interpolation
22	-0.131618	-6.79505e-08	interpolation
23	-0.131618	-2.98428e-13	interpolation
24	-0.131618	8.88178e-16	interpolation
25	-0.131618	8.88178e-16	interpolation

Zero found in the interval [-0.10949, -0.264]

a = -0.1316

每个迭代中当前子区间的端点列在标题 **a** 和 **b** 下，而端点处的相应 **humps** 值分别列在 **f(a)** 和 **f(b)** 下。

注意：端点 **a** 和 **b** 未按任何特定顺序列出：**a** 可能大于 **b** 或小于 **b**。

对于前 9 步，**humps** 的符号在当前子区间的两端点都为负号，如输出中所示。在第 10 步，**humps** 的符号在 **a** (-0.10949) 处为正号，但在 **b** (-0.264) 处为负号。从该点开始，如上一部分中所述，算法继续缩小区间 [-0.10949 -0.264]，直到它达到值 -0.1316。

另请参阅

详细信息

- “多项式的根”（第 6-4 页）
- “优化非线性函数”（第 9-2 页）
- “Systems of Nonlinear Equations”（Optimization Toolbox）

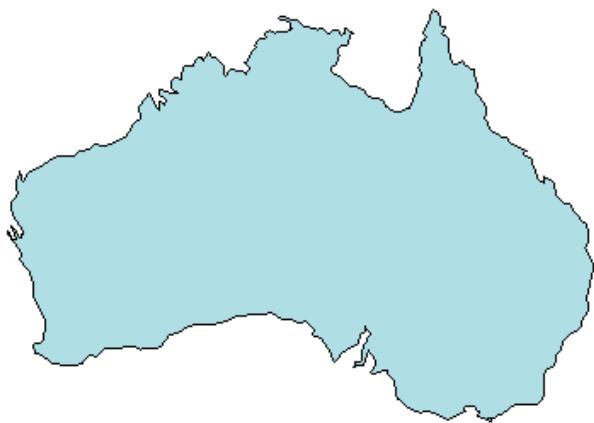
计算几何学

- “三角剖分表示法” (第 7-2 页)
- “使用 Delaunay 三角剖分” (第 7-13 页)
- “创建和编辑 Delaunay 三角剖分” (第 7-36 页)
- “空间搜索” (第 7-51 页)
- “Voronoi 图” (第 7-57 页)
- “区域边界的类型” (第 7-64 页)
- “计算凸包” (第 7-70 页)

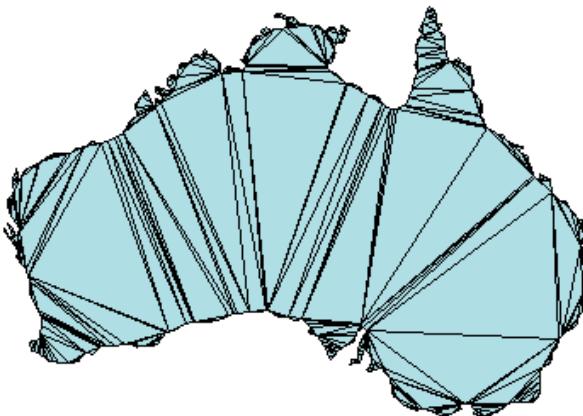
三角剖分表示法

二维和三维域

三角剖分通常用于表示计算机图形、物理建模、地理信息系统、医学成像及其他应用领域中的二维和三维几何域。如下所示的地图多边形

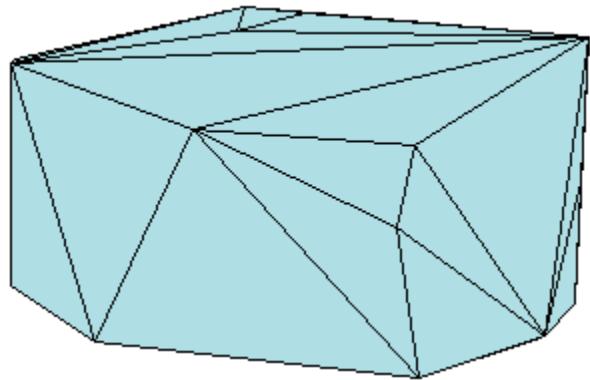


可以通过如下所示的地图的三角剖分来表示。



三角剖分将一个复杂的多边形分解为一组较简单的三角多边形。可以使用这些多边形开发基于几何学的算法或图形应用。

同样，可以使用三角剖分表示三维几何域的边界。下图显示了三维空间中一组点的凸包。凸包的每个分面都是三角形。

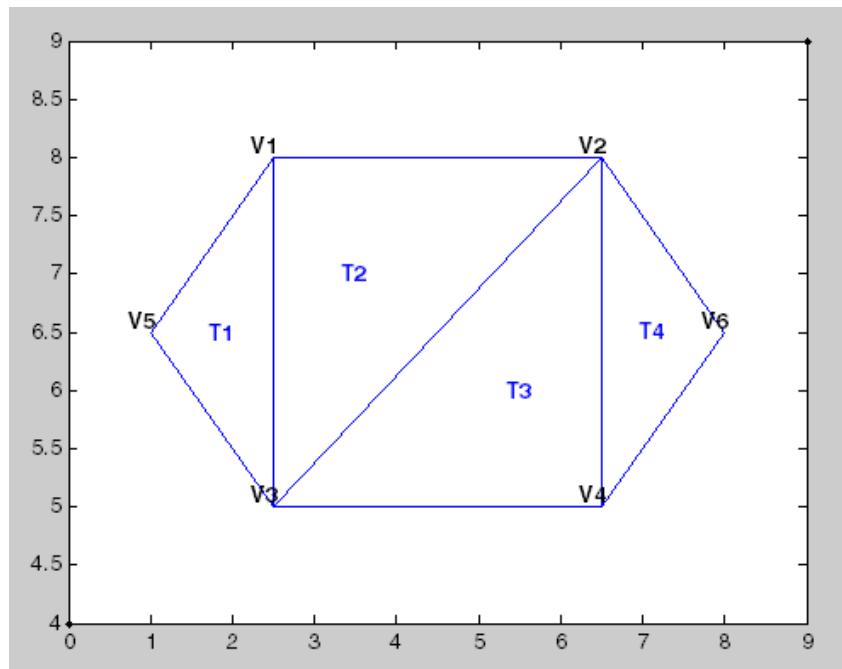


三角剖分矩阵格式

MATLAB 使用矩阵格式表示三角剖分。此格式包含以下两个部分：

- 采用一个矩阵表示的顶点，该矩阵中每行包含三角剖分中一个点的坐标。
- 采用一个矩阵表示的三角剖分连接，该矩阵中每行定义一个三角形或四面体。

此图显示了一个简单的二维三角剖分。



下表显示了顶点信息。

顶点		
顶点 ID	x 坐标	y 坐标
V1	2.5	8.0
V2	6.5	8.0

顶点		
V3	2.5	5.0
V4	6.5	5.0
V5	1.0	6.5
V6	8.0	6.5

上一个表中的数据在 MATLAB 环境中存储为矩阵。顶点 ID 是用于标识特定顶点的标签。这些标签用于说明顶点 ID 的概念，但不会显式存储。矩阵的行号充当顶点 ID。

该表中显示了三角剖分连接数据。

连接			
三角形 ID	边界顶点的 ID		
T1	5	3	1
T2	3	2	1
T3	3	4	2
T4	4	6	2

此表中的数据在 MATLAB 环境中存储为矩阵。三角形 ID 是用于标识特定三角形的标签。这些标签用于说明三角形 ID 的概念，但不会显式存储。矩阵的行号充当三角形 ID。

可以看到三角形 T1 是由三个顶点 {V5, V3, V1} 定义的。同样，T4 是由顶点 {V4, V6, V2} 定义的。此格式可自然扩展到要求更多数据列的更高维度。例如，三维空间中的四面体由四个顶点定义，每个顶点都包含三个坐标 (x, y, z)。

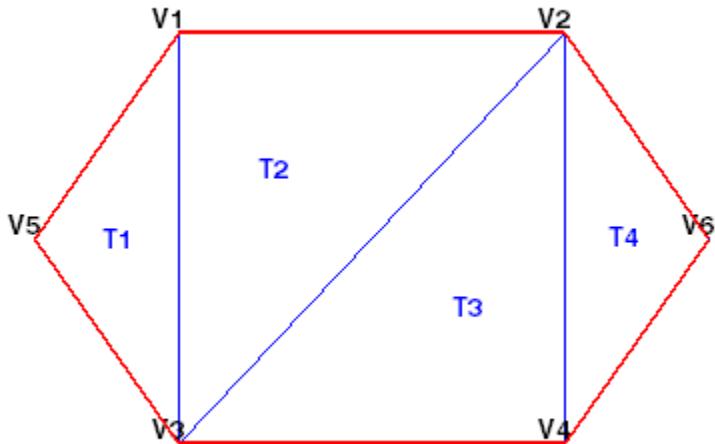
您可以使用 MATLAB 来表示和查询以下类型的三角剖分：

- 二维三角剖分，包含由顶点和边线限定的三角形
- 三维曲面三角剖分，包含由顶点和边线限定的三角形
- 三维三角剖分，包含由顶点、边线和面限定的四面体

使用三角剖分类查询三角剖分

矩阵格式提供基于紧凑型低级别数组的表示形式的三角剖分。使用三角剖分开发算法时，可能需要更多有关几何属性、拓扑和邻接信息的信息。

例如，在绘制如下所示的带注释的三角剖分之前，可以计算三角形内心。在这种情况下，使用内心显示每个三角形中的三角形标签 (T1、T2 等)。如果要用红色绘制边界，需要确定仅由一个三角形引用的边线。



三角剖分类

可以使用 `triangulation` 创建矩阵格式的任何二维或三维三角剖分数据的内存中表示形式，例如 `delaunay` 函数或其他软件工具的矩阵输出。使用 `triangulation` 表示数据时，可以执行拓扑和几何查询，使用这些查询可开发几何算法。例如，可以查找附加到顶点的三角形或四面体，这些三角形或四面体共享边线、外心和其他特征。

可以通过以下两种方式之一创建 `triangulation`：

- 将矩阵格式的现有数据传递到 `triangulation`。这些数据可以是 MATLAB 函数的输出，例如 `delaunay` 或 `convhull`。您可以导入通过其他软件应用程序创建的三角剖分数据。使用导入的数据时，请确保连接数据使用从 1 开始而不是从 0 开始的索引来引用顶点数组。
- 将一组点传递到 `delaunayTriangulation`。生成的 Delaunay 三角剖分一种特殊的 `triangulation`。这意味着可以对数据执行任何 `triangulation` 查询以及任何 Delaunay 特定的查询。在比较正式的 MATLAB 语言术语中，`delaunayTriangulation` 是 `triangulation` 的子类。

通过矩阵数据创建三角剖分

此示例说明如何使用三角剖分矩阵数据创建 `triangulation`，探索数据，以及可以对数据执行的操作。

创建包含顶点数据的矩阵 `P`。

```
P = [ 2.5  8.0
      6.5  8.0
      2.5  5.0
      6.5  5.0
      1.0  6.5
      8.0  6.5];
```

定义连接 `T`。

```
T = [5 3 1;
      3 2 1;
      3 4 2;
      4 6 2];
```

从该数据创建 **triangulation**。

```
TR = triangulation(T,P)
```

```
TR =
    triangulation with properties:
```

```
    Points: [6x2 double]
    ConnectivityList: [4x3 double]
```

按照访问 **struct** 的字段的相同方式访问 **triangulation** 中的属性。例如，检查包含顶点坐标的 **Points** 属性。

```
TR.Points
```

```
ans = 6×2
```

```
2.5000 8.0000
6.5000 8.0000
2.5000 5.0000
6.5000 5.0000
1.0000 6.5000
8.0000 6.5000
```

接下来检查连接。

```
TR.ConnectivityList
```

```
ans = 4×3
```

```
5 3 1
3 2 1
3 4 2
4 6 2
```

Points 和 **ConnectivityList** 属性定义三角剖分的矩阵数据。

triangulation 类是矩阵数据的封装类。真实益处是 **triangulation** 类方法的有用性。这些方法类似于接受 **triangulation** 和其他相关输入数据的函数。

triangulation 类提供了一种简单方法创建 **ConnectivityList** 属性矩阵的索引。访问三角剖分中的第一个三角形。

```
TR.ConnectivityList(1,:)
```

```
ans = 1×3
```

```
5 3 1
```

获取第一个三角形的另一种方法是 **TR(1,:)**。

检查第一个三角形的第一个顶点。

```
TR(1,1)
```

```
ans = 5
```

检查第一个三角形的第二个顶点。

```
TR(1,2)
```

```
ans = 3
```

现在检查三角剖分中的所有三角形。

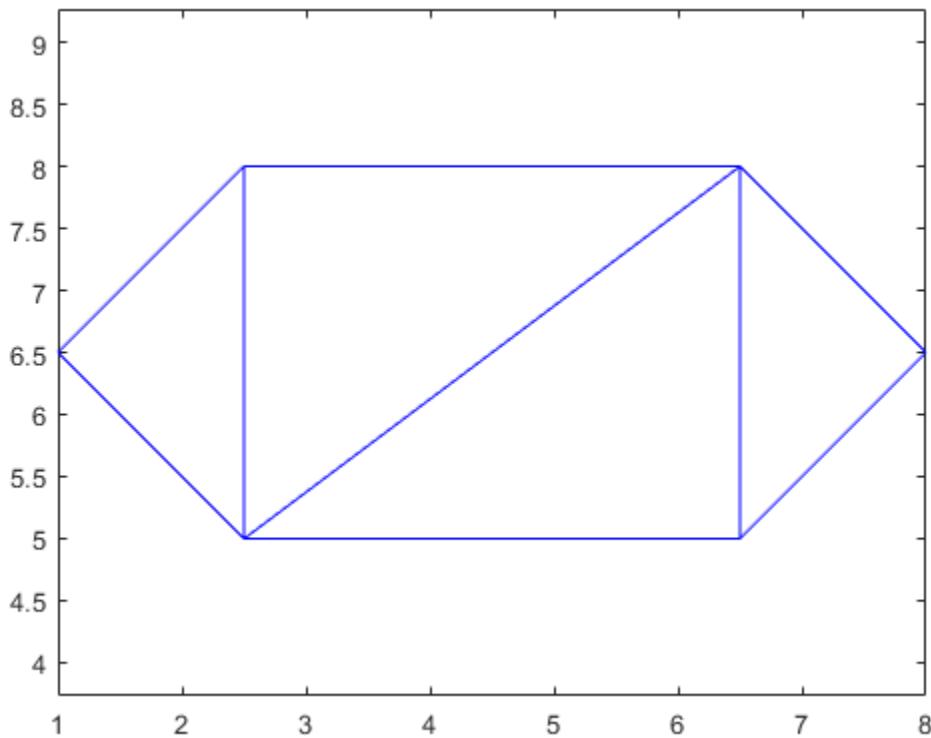
```
TR(:, :)
```

```
ans = 4×3
```

5	3	1
3	2	1
3	4	2
4	6	2

使用 `triplot` 绘制 `triangulation`。`triplot` 函数不是 `triangulation` 方法，但它接受并可绘制 `triangulation`。

```
figure
triplot(TR)
axis equal
```

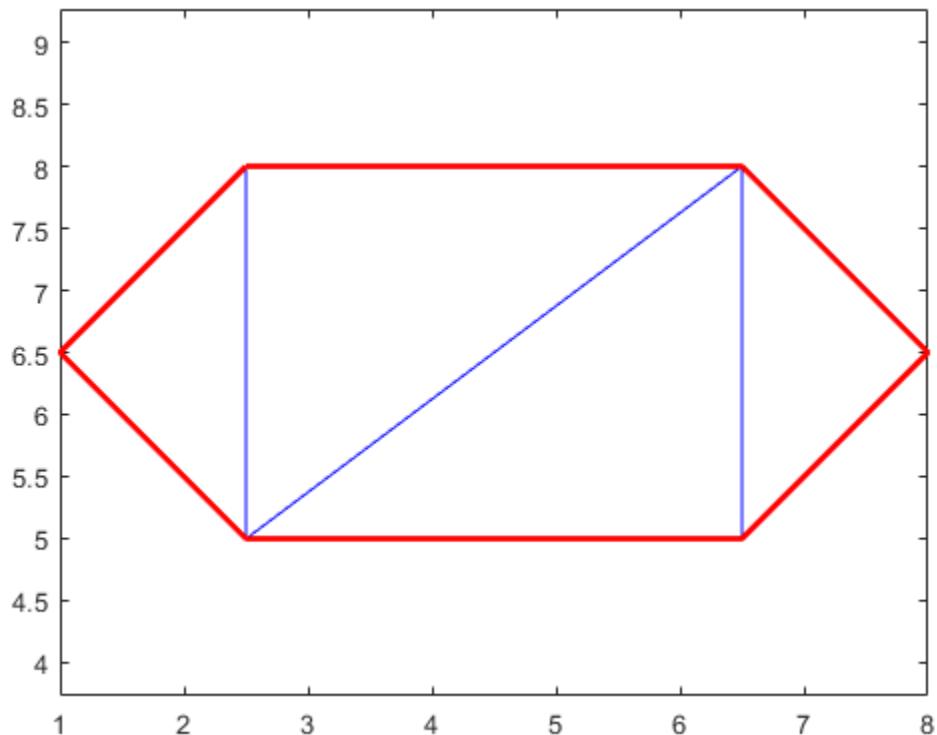


使用 `triangulation` 方法 `freeBoundary` 可查询自由边界并在绘图中突出显示该边界。该方法返回仅一个三角形共享的三角剖分边线。返回的边线以顶点 ID 方式表示。

```
boundaryedges = freeBoundary(TR);
```

现在用红色线绘制边界线。

```
hold on
plot(P(boundaryedges,1),P(boundaryedges,2),'r','LineWidth',2)
hold off
```



可以使用 `freeBoundary` 方法验证三角剖分。例如，如果观察三角剖分内部的红色边线，会发现它指明三角形的连接方式存在问题。

使用 `delaunayTriangulation` 创建三角剖分

此示例说明如何使用 `delaunayTriangulation` 创建 Delaunay 三角剖分。

使用 `delaunayTriangulation` 类创建 Delaunay 三角剖分时，会自动获得 `triangulation` 方法的访问权限，因为 `delaunayTriangulation` 是 `triangulation` 的子类。

基于一组点创建 `delaunayTriangulation`。

```
P = [ 2.5  8.0
      6.5  8.0
      2.5  5.0
      6.5  5.0
      1.0  6.5
      8.0  6.5];
```

```
DT = delaunayTriangulation(P)
```

```
DT =  
delaunayTriangulation with properties:
```

```
    Points: [6x2 double]  
ConnectivityList: [4x3 double]  
    Constraints: []
```

生成的 `delaunayTriangulation` 对象具有属性 `Points` 和 `ConnectivityList`, 就像 `triangulation` 对象一样。

使用直接索引访问三角剖分, 就像 `triangulation` 一样。例如, 可以检查第一个三角形的连接。

```
DT(1,:)
```

```
ans = 1×3
```

```
5 3 1
```

接下来, 检查整个三角剖分的连接。

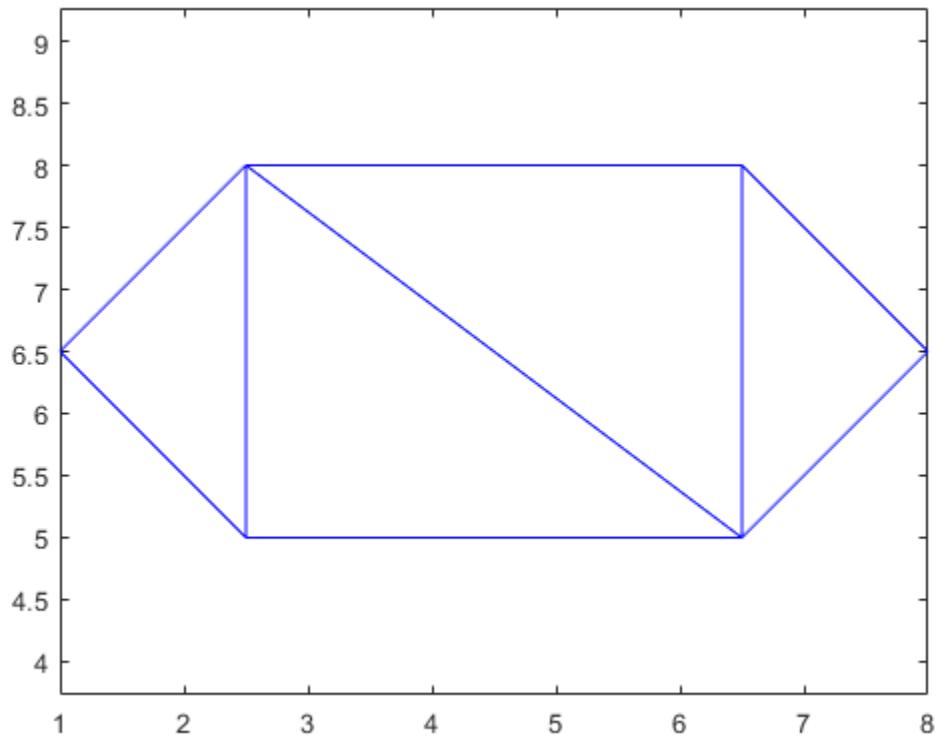
```
DT(:,1)
```

```
ans = 4×3
```

```
5 3 1  
3 4 1  
1 4 2  
4 6 2
```

使用 `triplot` 函数绘制三角剖分。

```
triplot(DT)  
axis equal
```



父类 triangulation 提供了 incenter 方法计算每个三角形的内心。

IC = incenter(DT)

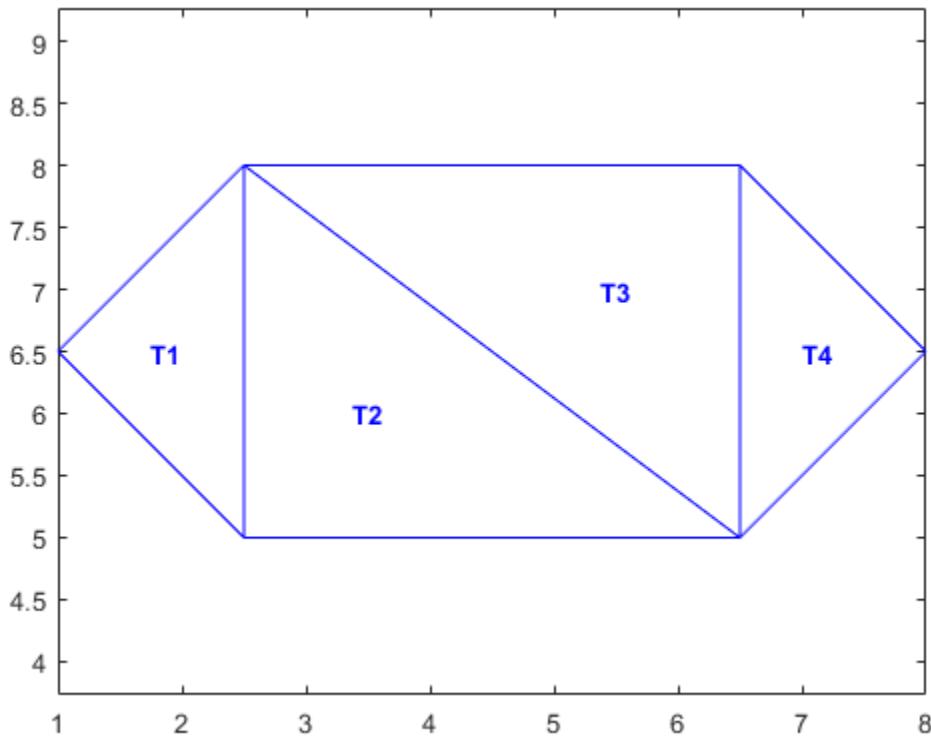
IC = 4×2

1.8787	6.5000
3.5000	6.0000
5.5000	7.0000
7.1213	6.5000

返回的值 IC 是表示三角形内心的坐标数组。

现在，使用内心找到将三角形标签放置于绘图中的位置。

```
hold on
numtri = size(DT,1);
trilabels = arrayfun(@(P) {sprintf('T%d', P)}, (1:numtri)');
Htl = text(IC(:,1),IC(:,2),trilabels,'FontWeight','bold',...
'HorizontalAlignment','center','Color','blue');
hold off
```



无需使用 **delaunayTriangulation** 创建 Delaunay 三角剖分，可以使用 **delaunay** 函数创建三角剖分连接数据，然后将连接数据传递到 **triangulation**。例如，

```
P = [ 2.5  8.0
      6.5  8.0
      2.5  5.0
      6.5  5.0
      1.0  6.5
      8.0  6.5];
```

```
T = delaunay(P);
TR = triangulation(T,P);
IC = incenter(TR);
```

两种方法在此示例中都有效，但如果要创建 Delaunay 三角剖分并对其进行查询，由于以下原因应使用 **delaunayTriangulation**：

- **delaunayTriangulation** 类可以提供适合处理三角剖分的其他方法。例如，可以执行最近邻点和三角形内的点搜索。
- 它允许您编辑三角剖分以添加、移动或删除点。
- 它允许您创建约束性的 Delaunay 三角剖分。这允许您创建二维域的三角剖分。

另请参阅

[delaunay](#) | [delaunayTriangulation](#) | [freeBoundary](#) | [triangulation](#) | [triplot](#)

详细信息

- “三角剖分表示法” (第 7-2 页)
- “使用 Delaunay 三角剖分” (第 7-13 页)
- “空间搜索” (第 7-51 页)

使用 Delaunay 三角剖分

本节内容

“Delaunay 三角剖分的定义”（第 7-13 页）

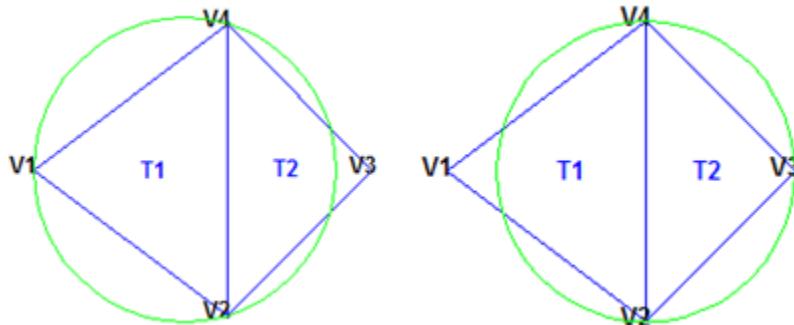
“创建 Delaunay 三角剖分”（第 7-14 页）

“包含重复位置的点集的三角剖分”（第 7-34 页）

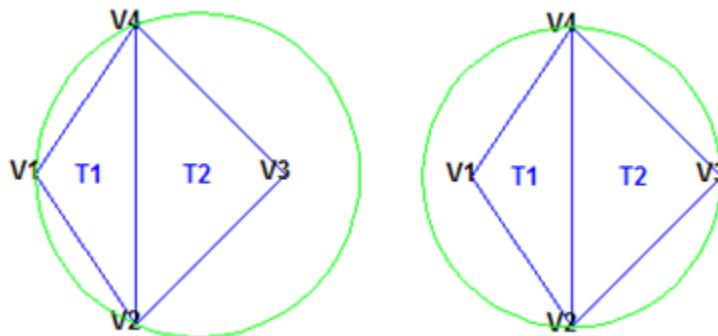
Delaunay 三角剖分的定义

Delaunay 三角剖分广泛应用于许多不同应用程序中的科学计算。虽然有大量的计算三角剖分的算法，但 Delaunay 三角剖分以其实用的几何属性广受欢迎。

基本属性是 Delaunay 规则。如果是二维三角剖分，通常将其称为空外接圆规则。对于一组二维点而言，这些点的 Delaunay 三角剖分可确保与每个三角形相关的外接圆的内部都不包含其他点。此属性非常重 要。在下面的插图中，与 T1 关联的外接圆是空的。该外接圆的内部不包含任何点。与 T2 相关的外接圆为 空。该外接圆的内部不包含任何点。这种三角剖分便是 Delaunay 三角剖分。

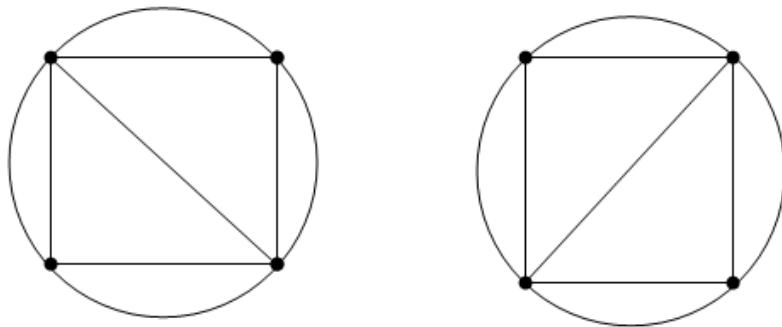


下面的三角形则不同。与 T1 关联的外接圆不为空。它的内部包含 V3。与 T2 关联的外接圆不为空。它的 内部包含 V1。这种三角剖分不是 Delaunay 三角剖分。

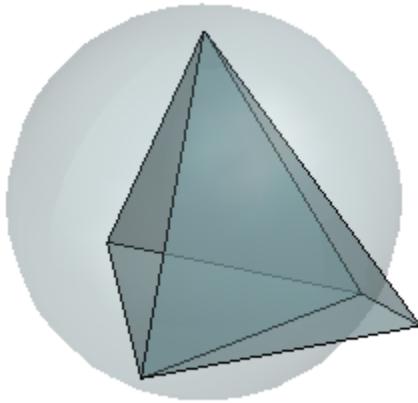


Delaunay 三角剖分堪称“外形整齐”，原因在于为满足空外接圆属性，优先选择带有较大内角的三角形，而不是带有较小内角的三角形。非 Delaunay 三角剖分中的三角形在顶点 V2 和 V4 处呈锐角。如果将 {V2, V4} 边替换为连接 V1 和 V3 的边，会实现最小角的最大化并且使得该三角剖分变为 Delaunay 三角 剖分。另外，Delaunay 三角剖分将最近邻点的点连接在一起。这两个特征（外形整齐和最近邻点关系）在 实践中具有重要的作用，有助于促进在散点数据插值中使用 Delaunay 三角剖分。

虽然 Delaunay 属性定义明确，但存在退化点集时三角剖分的拓扑并不唯一。在二维中，4 个或更多特征点位于同一圆中时会引发退化。例如，正方形的顶点不具有唯一的 Delaunay 三角剖分。



Delaunay 三角剖分的属性扩展到更高的维度。三维点集的三角剖分由四面体组成。下图显示了一个简单的由 2 个四面体组成的三维 Delaunay 三角剖分。显示一个四面体的外接球以突出显示空外接球规则。



三维 Delaunay 三角剖分可生成符合空外接球规则的四面体。

创建 Delaunay 三角剖分

MATLAB 提供两种创建 Delaunay 三角剖分的方法：

- 函数 `delaunay` 和 `delaunayn`
- `delaunayTriangulation` 类

`delaunay` 函数支持创建二维和三维 Delaunay 三角剖分。`delaunayn` 函数支持创建四维或更高维度的 Delaunay 三角剖分。

提示 对于中型至大型点集而言，由于所需的内存呈指数级增长，创建六维以上的 Delaunay 三角剖分通常不太现实。

`delaunayTriangulation` 类支持创建二维和三维 Delaunay 三角剖分。它提供了多种适用于开发基于三角剖分的算法的方法。这些类方法与函数相似，但它们仅限于处理使用 `delaunayTriangulation` 创建的

三角剖分。**delaunayTriangulation** 类还支持创建相关构造，例如凸包和 Voronoi 图。它还支持创建约束性 Delaunay 三角剖分。

总的说来：

- 在仅需要基本三角剖分数据并且数据对应用程序已足够完整时，**delaunay** 函数非常有用。
- **delaunayTriangulation** 类提供了更多功能用于开发基于三角剖分的应用程序。在需要三角剖分并且希望执行以下任意操作时，该类非常有用：
 - 在三角剖分中搜索包含查询点的三角形或四面体。
 - 使用三角剖分执行最近邻点搜索。
 - 查询三角剖分的拓扑邻接或几何属性。
 - 修改三角剖分以插入或删除点。
 - 约束三角剖分中的边界 - 这称为约束性的 Delaunay 三角剖分。
 - 对多边形进行三角剖分并（可选）删除域外部的三角形。
 - 使用 Delaunay 三角剖分计算凸包或 Voronoi 图。

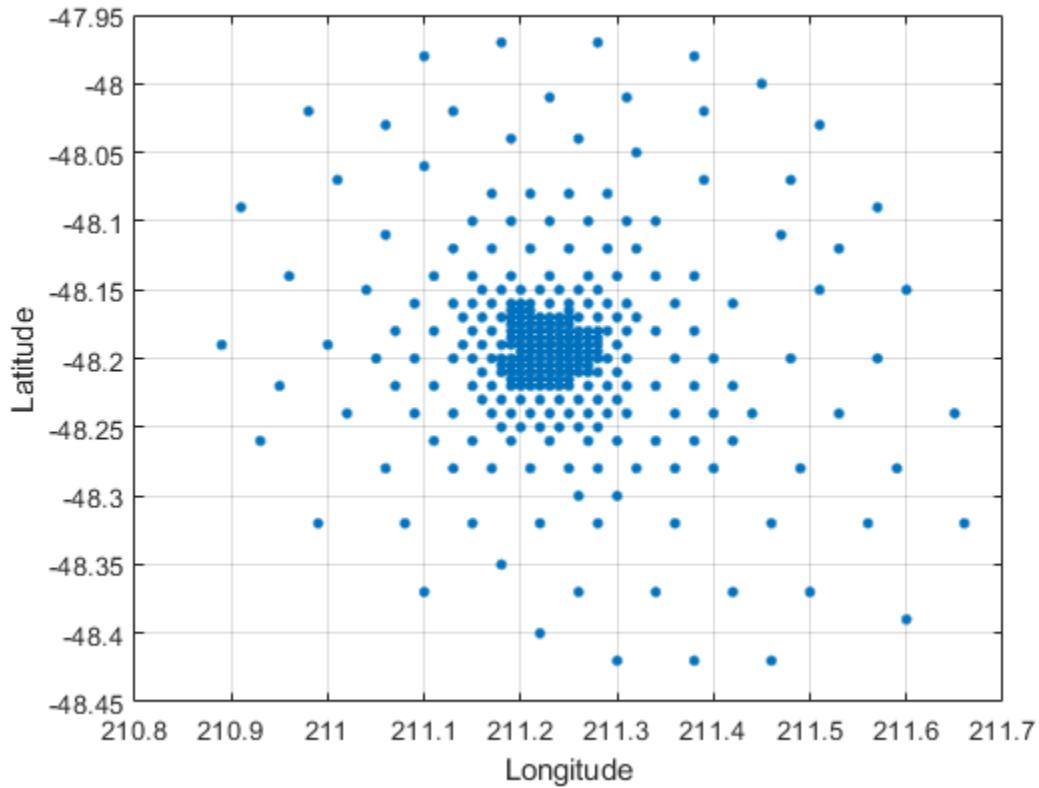
使用 **delaunay** 和 **delaunayn** 函数

delaunay 和 **delaunayn** 函数获取点集并生成矩阵格式的三角剖分。请参阅“三角剖分矩阵格式”（第 7-3 页）以了解有关该数据结构体的详细信息。在二维模式下，**delaunay** 函数通常用于生成此类三角剖分，即它可用于绘制根据一组分散数据点来定义的曲面。在该应用中，必须注意该方法仅在曲面为单值型时可用。例如，它不能用于绘制球面，因为有两个 z 值对应于同一个 (x, y) 坐标。通过一个简单示例来演示 **delaunay** 函数如何可用于绘制表示抽样数据集的曲面。

此示例说明如何使用 **delaunay** 函数创建海底山数据集的二维 Delaunay 三角剖分。海底山是指水下山脉。该数据集包含一组经度 (x) 和纬度 (y) 位置，以及在这些坐标测量的对应海底山海拔 (z)。

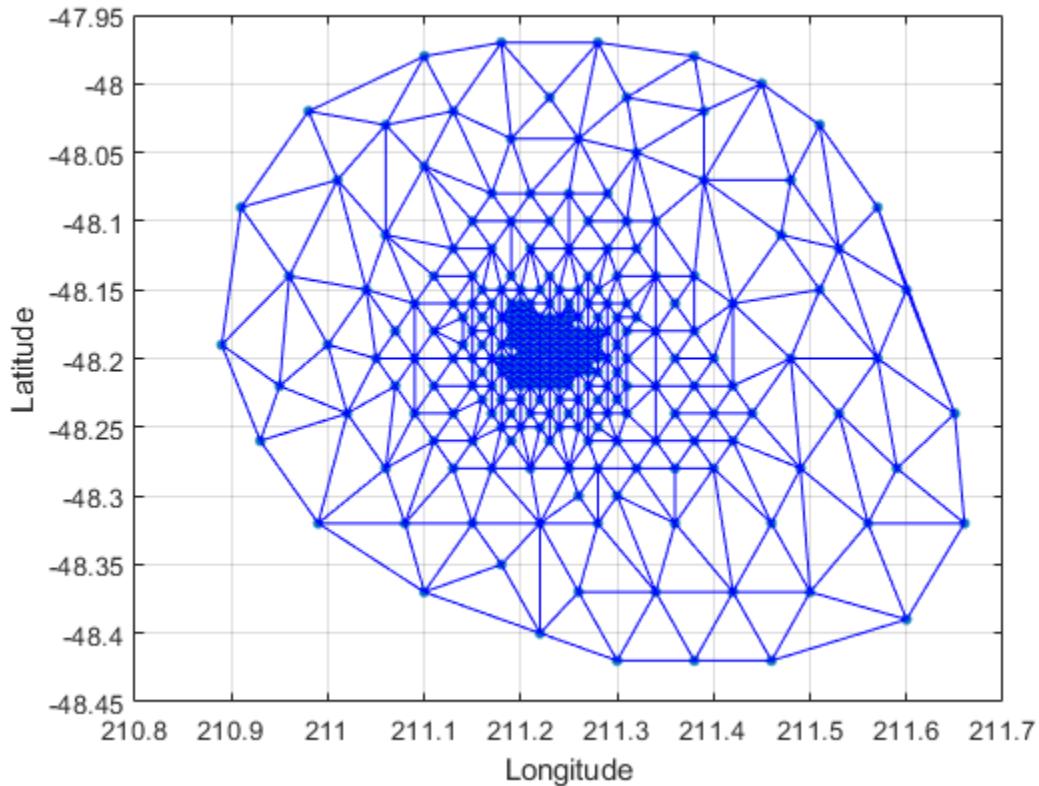
加载海底山数据集并以散点图的方式查看 (x, y) 数据。

```
load seamount
plot(x,y,'','markersize',12)
xlabel('Longitude'), ylabel('Latitude')
grid on
```



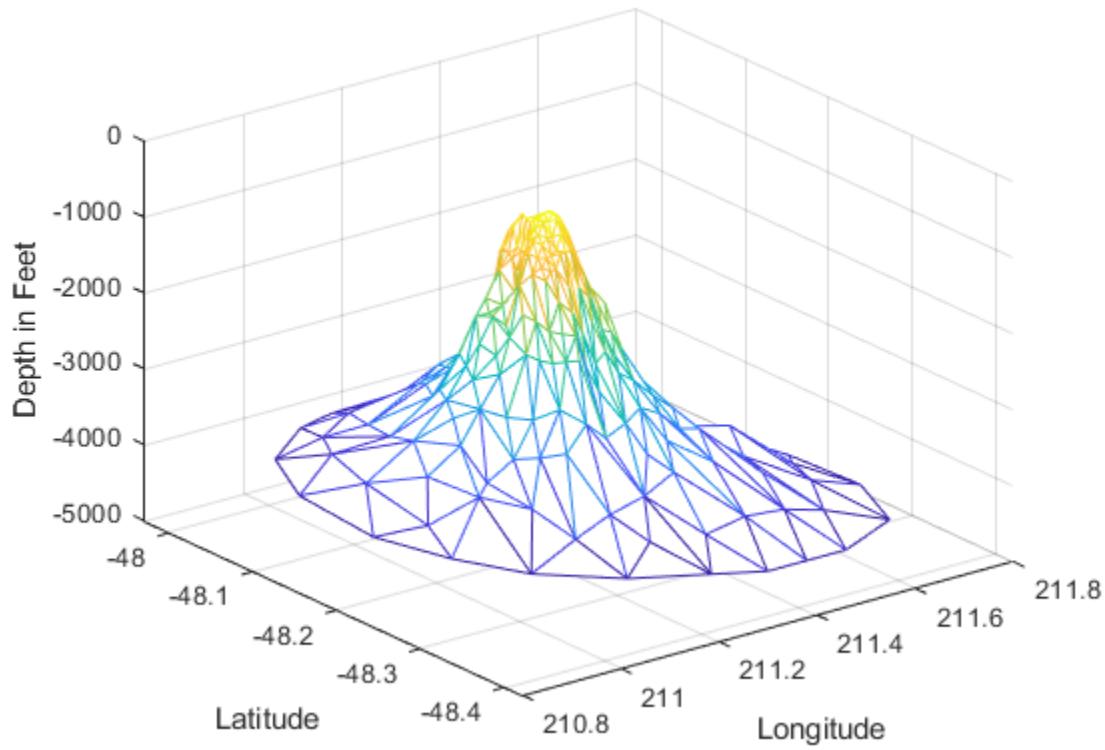
从该点集构建 Delaunay 三角剖分，并使用 `triplot` 在现有图窗中绘制三角剖分。

```
tri = delaunay(x,y);
hold on, triplot(tri,x,y), hold off
```



添加海底山的深度数据 (z) 以提高顶点并创建曲面。创建新图窗并在线框模式下使用 trimesh 绘制曲面。

```
figure  
hidden on  
trimesh(tri,x,y,z)  
xlabel('Longitude'), ylabel('Latitude'), zlabel('Depth in Feet');
```

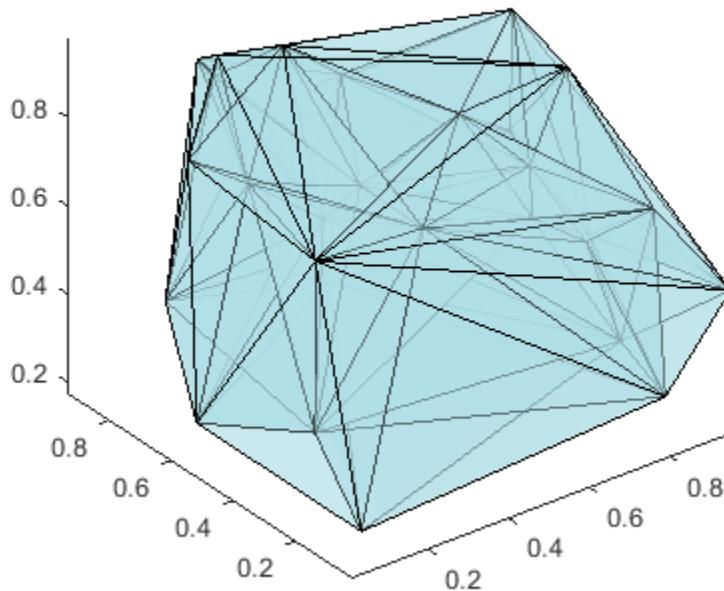


如果想要在着色模式下绘制该曲面，请使用 `trisurf` 而不是 `trimesh`。

也可以使用 `delaunay` 函数来创建三维 Delaunay 三角剖分。此三角剖分由四面体构成。

此示例说明如何创建随机数据集的三维 Delaunay 三角剖分。该三角剖分使用 `tetramesh` 进行绘图，`FaceAlpha` 选项增加了绘图的透明度。

```
X = gallery('uniformdata',[30 3],0);
tet = delaunay(X);
faceColor = [0.6875 0.8750 0.8984];
tetramesh(tet,X,'FaceColor', faceColor,'FaceAlpha',0.3);
```



MATLAB 提供 `delaunayn` 函数以支持创建四维及更高维的 Delaunay 三角剖分。此外，还提供了两个补充函数 `tsearchn` 和 `dsearchn`，以支持对 N 维三角剖分执行空间搜索。有关基于三角剖分的搜索的详细信息，请参阅“空间搜索”（第 7-51 页）。

使用 `delaunayTriangulation` 类

`delaunayTriangulation` 类提供另一种在 MATLAB 中创建 Delaunay 三角剖分的方法。虽然 `delaunay` 和 `delaunayTriangulation` 使用相同的基本算法并生成相同的三角剖分，但 `delaunayTriangulation` 提供了适用于开发基于 Delaunay 的算法的补充方法。这些方法与函数类似，可以将它们与三角剖分数据一起打包到称为类的容器中。将所有内容集中在类中，这样可以提供更加有条理的结构，从而提高易用性。它还可以提高基于三角剖分的搜索的性能，例如点位置和最近邻位置。`delaunayTriangulation` 支持对 Delaunay 三角剖分进行增量编辑。还可以实行二维中的边约束。

“三角剖分表示法”（第 7-2 页）引入了 `triangulation` 类，该类支持对二维和三维三角剖分进行拓扑和几何查询。`delaunayTriangulation` 是一种特殊的 `triangulation`。这意味着除了可以对 `delaunayTriangulation` 执行 Delaunay 特定的查询外，还可以执行任何 `triangulation` 查询。在比较正式的 MATLAB 语言术语中，`delaunayTriangulation` 是 `triangulation` 的子类。

此示例说明如何使用 `delaunayTriangulation` 创建、查询和编辑 `seamount` 数据的 Delaunay 三角剖分。海底山数据集包含用于定义海拔山曲面的 (x, y) 位置和对应的海拔 (z)。

加载 (x, y) 数据并进行三角剖分。

```
load seamount
DT = delaunayTriangulation(x,y)
```

```
DT =
delaunayTriangulation with properties:
```

```
    Points: [294x2 double]
    ConnectivityList: [566x3 double]
    Constraints: []
```

由于没有施加任何边约束，因此 **Constraints** 属性为空。**Points** 属性表示顶点坐标，**ConnectivityList** 属性表示三角形。这两个属性共同定义了三角剖分的矩阵数据。

delaunayTriangulation 类是矩阵数据的封装类，并且提供了一组补充方法。可以按照访问结构体字段的方式访问 **delaunayTriangulation** 中的属性。

访问顶点数据。

```
DT.Points;
```

访问连接数据。

```
DT.ConnectivityList;
```

访问 **ConnectivityList** 属性中的第一个三角形。

```
DT.ConnectivityList(1,:)
```

```
ans = 1×3
```

```
205 230 262
```

delaunayTriangulation 提供了一种对 **ConnectivityList** 属性矩阵进行索引的简单方法。

访问第一个三角形。

```
DT(1,:)
```

```
ans = 1×3
```

```
205 230 262
```

检查第一个三角形的第一个顶点。

```
DT(1,1)
```

```
ans = 205
```

检查三角剖分中的所有三角形。

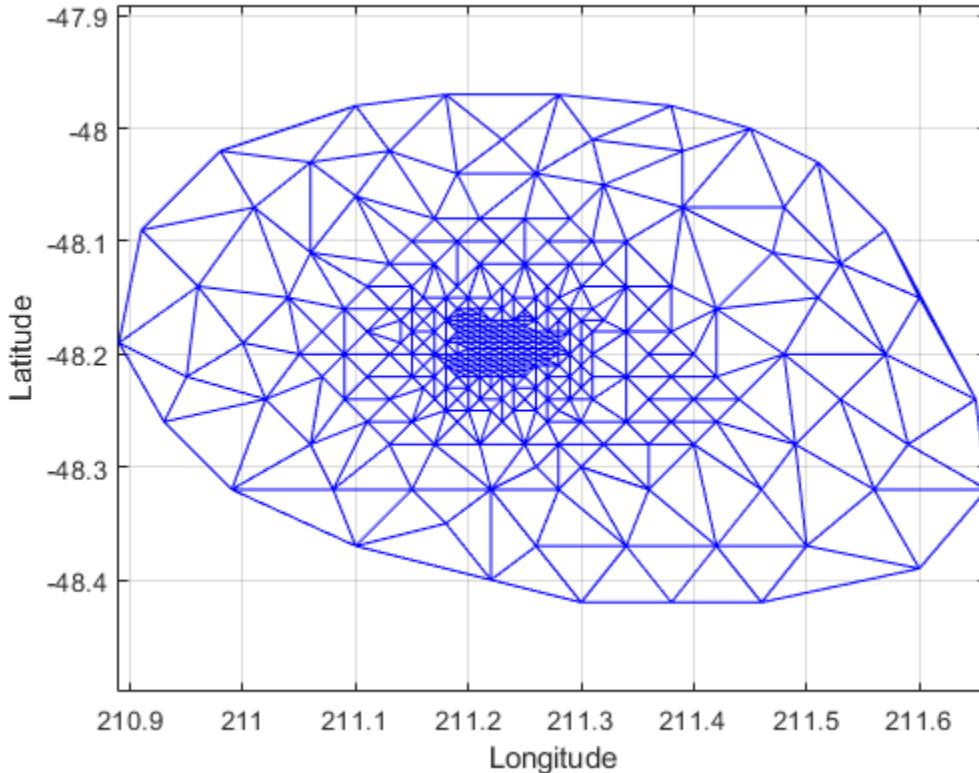
```
DT(:, :);
```

对 **delaunayTriangulation** 输出 **DT** 进行索引，这与对 **delaunay** 的三角剖分数组输出进行索引类似。二者的区别是您可以对 **DT** 调用额外方法（例如 **nearestNeighbor** 和 **pointLocation**）。

使用 **triplot** 绘制 **delaunayTriangulation**。**triplot** 函数不是 **delaunayTriangulation** 方法，但它接受并可绘制 **delaunayTriangulation**。

```
triplot(DT);
axis equal
```

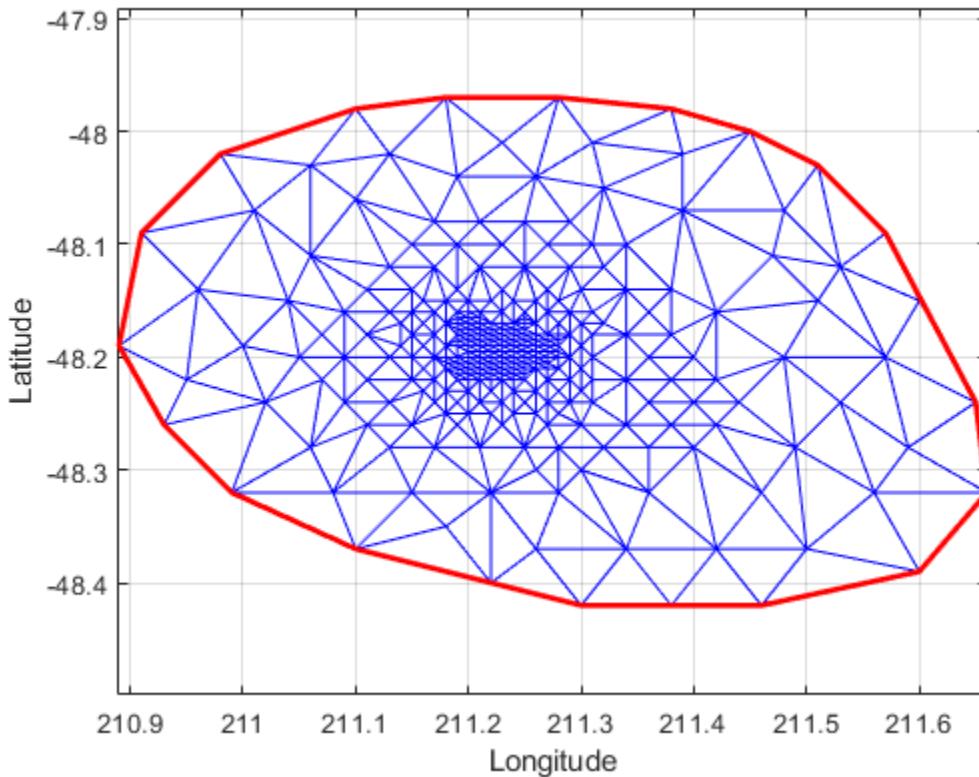
```
xlabel('Longitude'), ylabel('Latitude')
grid on
```



或者，可以使用 `tripplot(DT(:,:,1), DT.Points(:,1), DT.Points(:,2))`; 获取相同的绘图。

使用 `delaunayTriangulation` 方法 `convexHull` 计算凸包并将其添加到绘图。由于您已经有一个 Delaunay 三角剖分，因此可利用此方法来获得凸包，效率比使用 `convhull` 的完全计算方法更高。

```
hold on
k = convexHull(DT);
xHull = DT.Points(k,1);
yHull = DT.Points(k,2);
plot(xHull,yHull,'r','LineWidth',2);
hold off
```



您可以按增量方式编辑 `delaunayTriangulation`, 以添加或删除点。如果需要将点添加到现有的三角剖分, 增量添加方法的速度要快于对增广点集进行完整的重新三角剖分。当要删除的点数量相对现有的点数量较小时, 增量点删除的效率更高。

编辑三角剖分, 删除上一次计算的凸包上的点。

```

figure
plot(xHull,yHull,'r','LineWidth',2);
axis equal
xlabel('Longitude'),ylabel('Latitude')
grid on

% The convex hull topology duplicates the start and end vertex.
% Remove the duplicate entry.
k(end) = [];

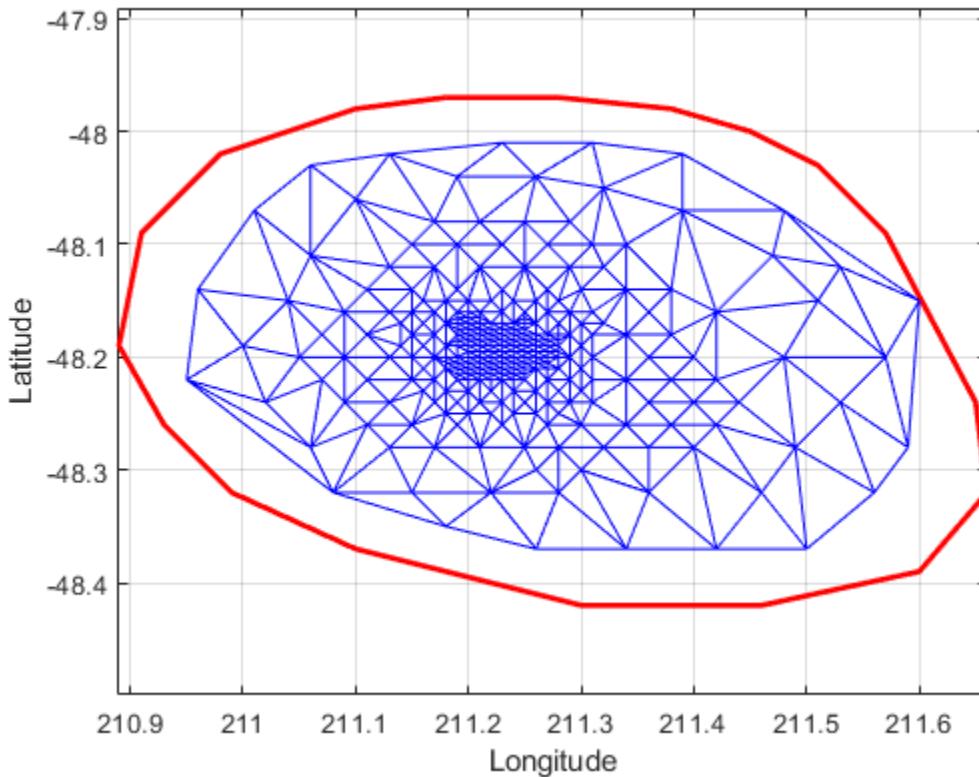
% Now remove the points on the convex hull.
DT.Points(k,:) = []

DT =
delaunayTriangulation with properties:

    Points: [274x2 double]
    ConnectivityList: [528x3 double]
    Constraints: []

```

```
% Plot the new triangulation.
hold on
triplot(DT);
hold off
```



紧靠凸包边界内侧有一个未删除的顶点。在图窗中使用放大工具可以看到，它确实处于凸包以内。您可以绘制顶点标签，以确定此顶点的索引，并从三角剖分中将其删除。或者，可以使用 **nearestNeighbor** 方法更轻松地确定索引。

该点靠近位置 (211.6, -48.15)。使用 **nearestNeighbor** 方法求最近的顶点。

```
vertexId = nearestNeighbor(DT, 211.6, -48.15)
```

```
vertexId = 50
```

现在从三角剖分中删除该顶点。

```
DT.Points(vertexId,:) = []
```

```
DT =
delaunayTriangulation with properties:
```

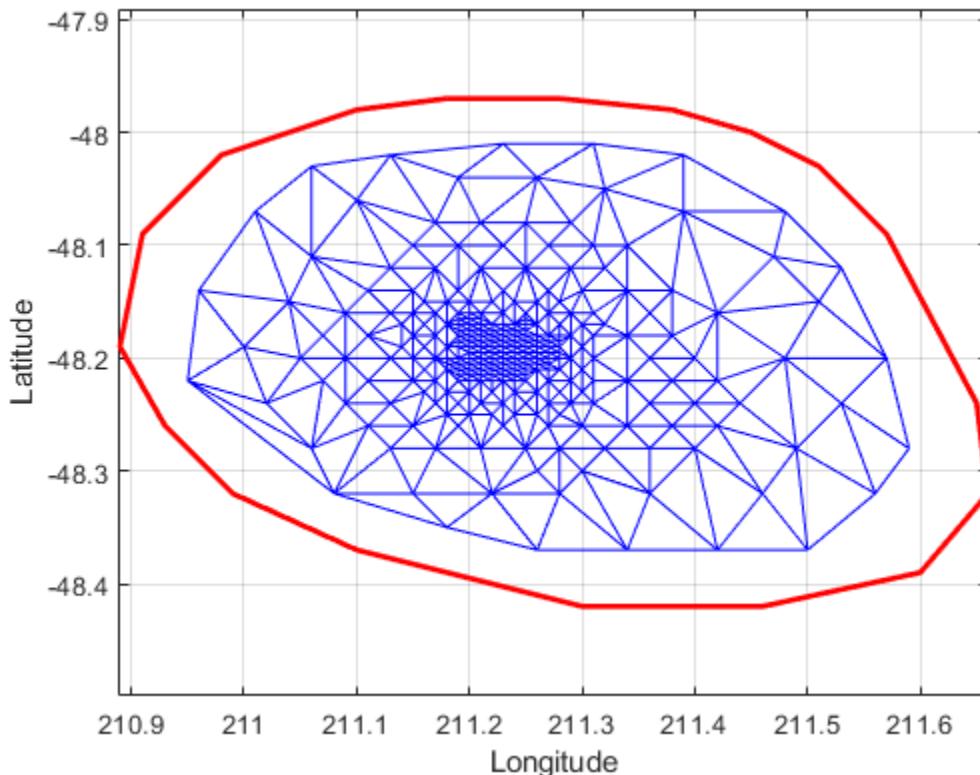
```
    Points: [273x2 double]
    ConnectivityList: [525x3 double]
    Constraints: []
```

对新的三角剖分进行绘图。

```

figure
plot(xHull,yHull,'r','LineWidth',2);
axis equal
xlabel('Longitude'),ylabel('Latitude')
grid on
hold on
triplot(DT);
hold off

```



将点添加到现有的三角剖分。添加 4 个点，在三角剖分的周围构成一个矩形。

```

Padditional = [210.9 -48.5; 211.6 -48.5; ...
    211.6 -47.9; 210.9 -47.9];
DT.Points(end+(1:4),:) = Padditional

```

```

DT =
delaunayTriangulation with properties:

    Points: [277x2 double]
    ConnectivityList: [548x3 double]
    Constraints: []

```

关闭所有现有图窗。

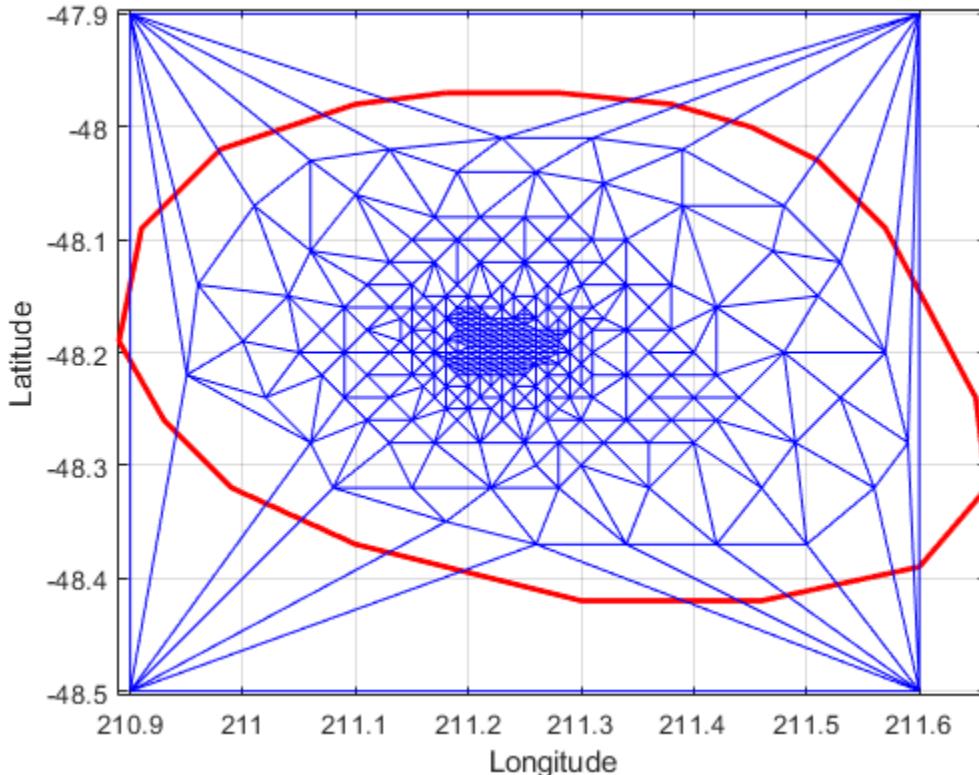
```
close all
```

对新的三角剖分进行绘图。

```

figure
plot(xHull,yHull,'r','LineWidth',2);
axis equal
xlabel('Longitude'),ylabel('Latitude')
grid on
hold on
triplot(DT);
hold off

```



您可以编辑三角剖分中的点，将其移到新位置。编辑附加点集的第一个点（顶点 ID 274）。

```
DT.Points(274,:) = [211 -48.4];
```

关闭所有现有图窗。

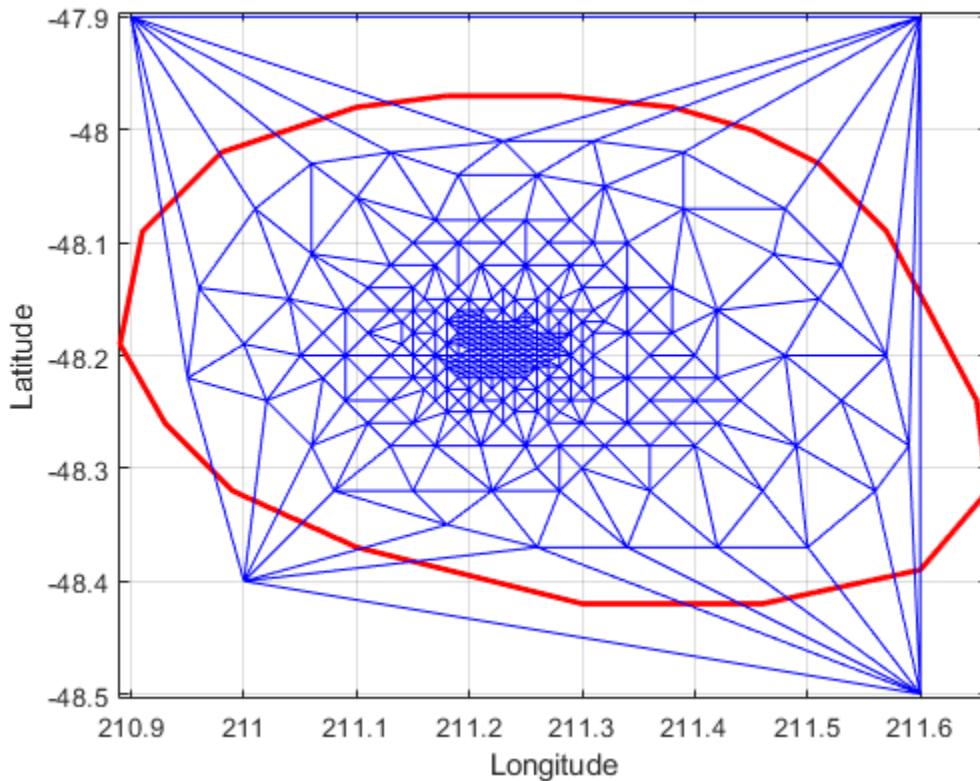
```
close all
```

对新的三角剖分进行绘图

```

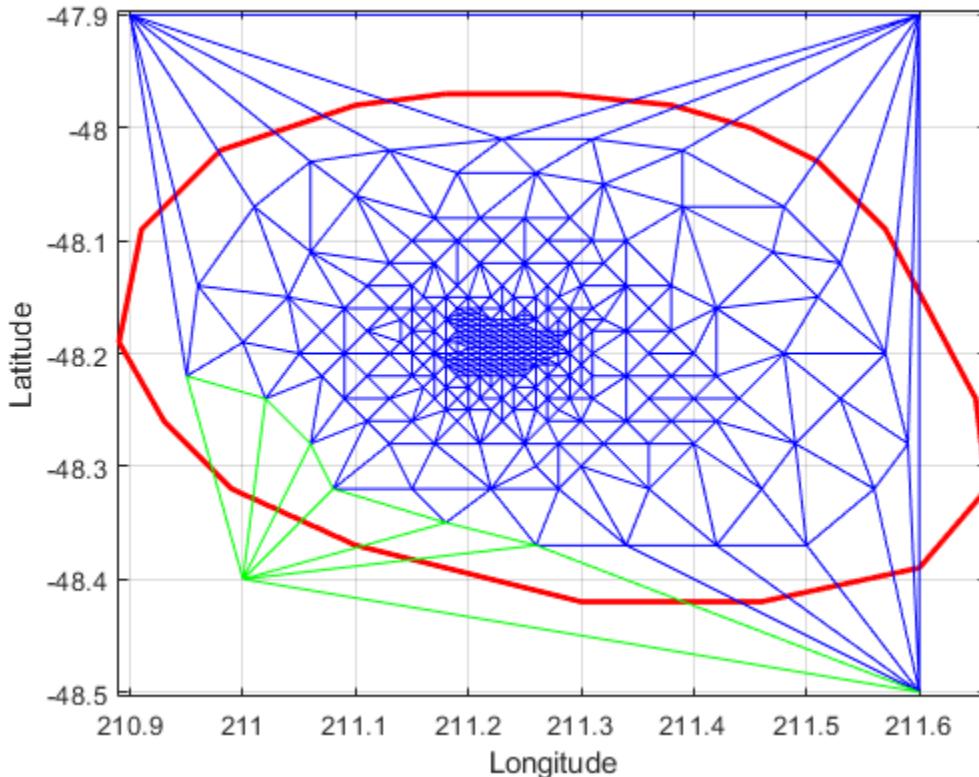
figure
plot(xHull,yHull,'r','LineWidth',2);
axis equal
xlabel('Longitude'),ylabel('Latitude')
grid on
hold on
triplot(DT);
hold off

```



使用 `triangulation` 类的方法 `vertexAttachments` 求连接的三角形。由于连接到顶点的三角形数量是可变的，因此该方法将在元胞数组中返回连接的三角形 ID。您需要使用括号提取内容。

```
attTris = vertexAttachments(DT,274);
hold on
triplot(DT(attTris{:,1}),DT.Points(:,1),DT.Points(:,2),'g')
hold off
```



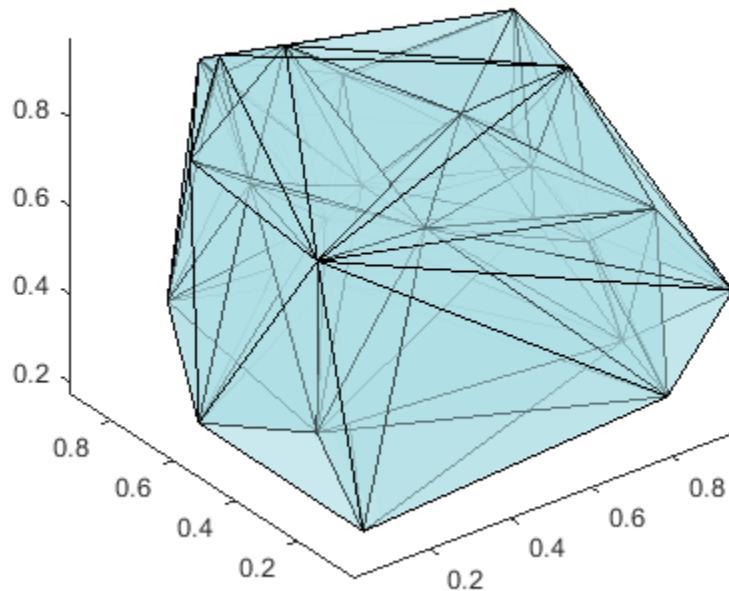
`delaunayTriangulation` 也可以用于对三维空间中的点进行三角剖分。生成的三角剖分由四面体构成。此示例说明如何使用 `delaunayTriangulation` 创建并绘制三维点的三角剖分。

```
P = gallery('uniformdata',30,3,0);
DT = delaunayTriangulation(P)

DT =
delaunayTriangulation with properties:
```

```
    Points: [30x3 double]
    ConnectivityList: [117x4 double]
    Constraints: []
```

```
faceColor = [0.6875 0.8750 0.8984];
tetramesh(DT,'FaceColor', faceColor, 'FaceAlpha',0.3);
```

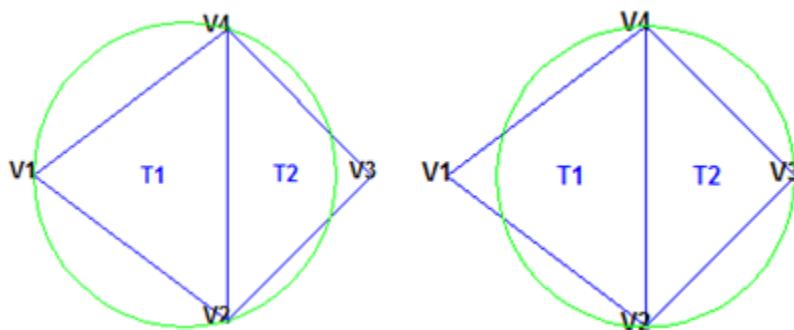


tetramesh 函数会同时绘制三角剖分的内面和外面。对于大型三维三角剖分，绘制内面可能导致不必要的资源使用。绘制边界可能更加适宜。您可以使用 **freeBoundary** 方法获取矩阵格式的边界三角剖分。然后将结果传递给 **trimesh** 或 **trisurf**。

受约束的 Delaunay 三角剖分

delaunayTriangulation 类允许您约束二维三角剖分中的边界。这意味着可以选择三角剖分中的一对点并约束用于连接这些点的边。可以将其作为“施加压力”绘制一对或多对点之间的边界。以下示例说明了边约束对三角剖分的影响。

下面的三角剖分是 Delaunay 三角剖分，因为它符合空外接圆规则。



使用指定的顶点 V1 和 V3 间的边约束对点集进行三角剖分。

定义点集。

```
P = [2 4; 6 1; 9 4; 6 7];
```

定义 V1 与 V3 之间的约束 C。

```
C = [1 3];
DT = delaunayTriangulation(P,C);
```

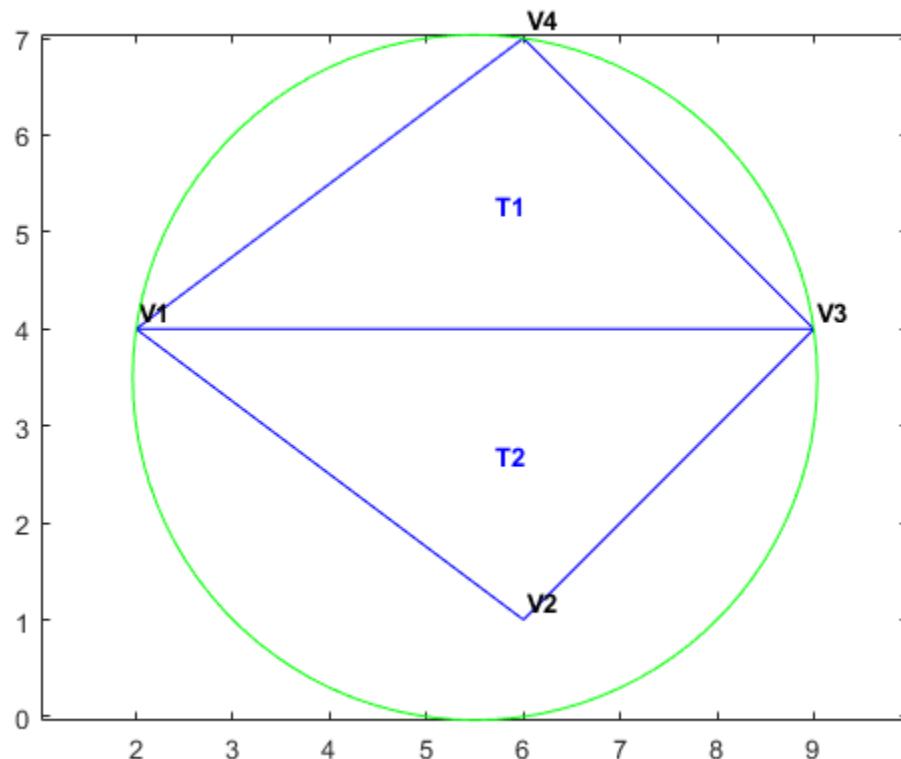
对三角剖分绘图并添加注释。

```
triplot(DT)
```

```
% Label the vertices.
hold on
numvx = size(P,1);
vxlabels = arrayfun(@(n) {sprintf('V%d', n)}, (1:numvx)');
Hpl = text(P(:,1)+0.2, P(:,2)+0.2, vxlabels, 'FontWeight', ...
'bold', 'HorizontalAlignment','center', 'BackgroundColor', ...
'none');
hold off

% Use the incenters to find the positions for placing triangle labels on the plot.
hold on
IC = incenter(DT);
numtri = size(DT,1);
trilabels = arrayfun(@(P) {sprintf('T%d', P)}, (1:numtri)');
Htl = text(IC(:,1),IC(:,2),trilabels,'FontWeight','bold', ...
'HorizontalAlignment','center','Color','blue');
hold off

% Plot the circumcircle associated with the triangle, T1.
hold on
[CC,r] = circumcenter(DT);
theta = 0:pi/50:2*pi;
xunit = r(1)*cos(theta) + CC(1,1);
yunit = r(1)*sin(theta) + CC(1,2);
plot(xunit,yunit,'g');
axis equal
hold off
```



已遵循顶点 (V1、V3) 之间的约束，但导致 Delaunay 规则失效。此外还导致 Delaunay 三角剖分固有的最近邻关系失效。这意味着如果三角剖分包含约束，则无法支持 delaunayTriangulation 提供的 nearestNeighbor 搜索方法。

在典型应用中，三角剖分可能由大量点构成，并且三角剖分中受到约束的边可能相对较少。此类三角剖分被视为局部非 Delaunay 三角剖分，因为三角剖分中的许多三角形可能遵循 Delaunay 规则，但在局部可能存在一些不遵循该规则的三角形。在许多应用中，空外接圆属性的局部松弛不成问题。

受约束的三角剖分通常用于对非凸多边形进行三角剖分。这些约束在多边形边与三角剖分边间建立了对应关系。利用此关系，可以提取表示该区域的三角剖分。以下示例说明如何使用受约束的 delaunayTriangulation 对非凸多边形进行三角剖分。

定义并绘制多边形。

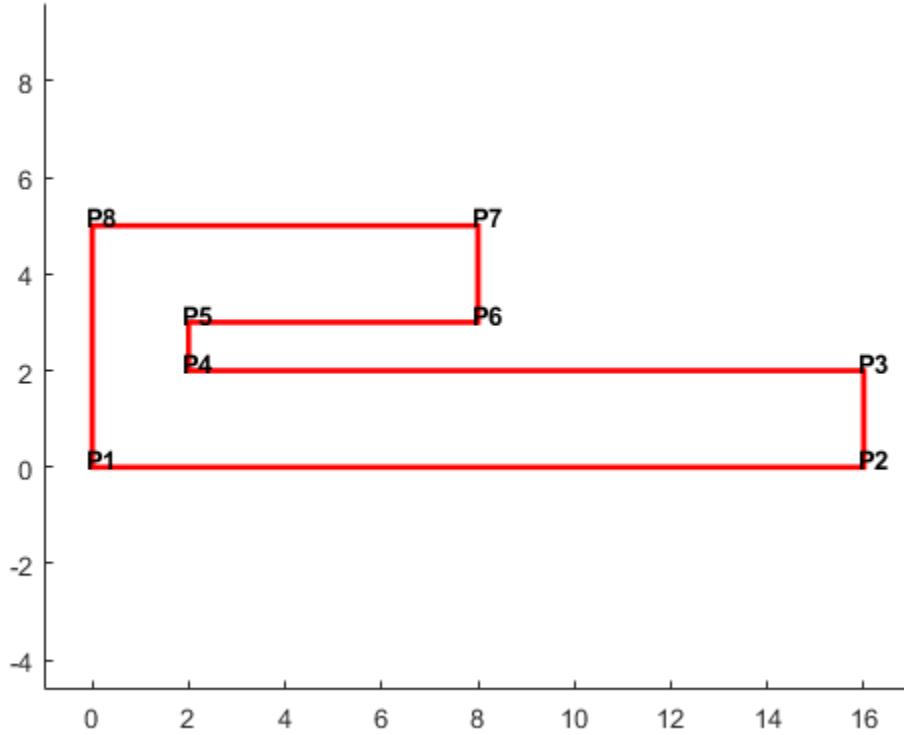
```

figure()
axis([-1 17 -1 6]);
axis equal
P = [0 0; 16 0; 16 2; 2 2; 2 3; 8 3; 8 5; 0 5];
patch(P(:,1),P(:,2),'r','LineWidth',2,'FaceColor',...
    'none','EdgeColor','r');

% Label the points.
hold on
numvtx = size(P,1);
vxlabels = arrayfun(@(n) {sprintf('P%d', n)}, (1:numvtx));
Hpl = text(P(:,1)+0.2, P(:,2)+0.2, vxlabels, 'FontWeight',...
    'bold', 'HorizontalAlignment','center', 'BackgroundColor',...
    'white');

```

```
'none');
hold off
```

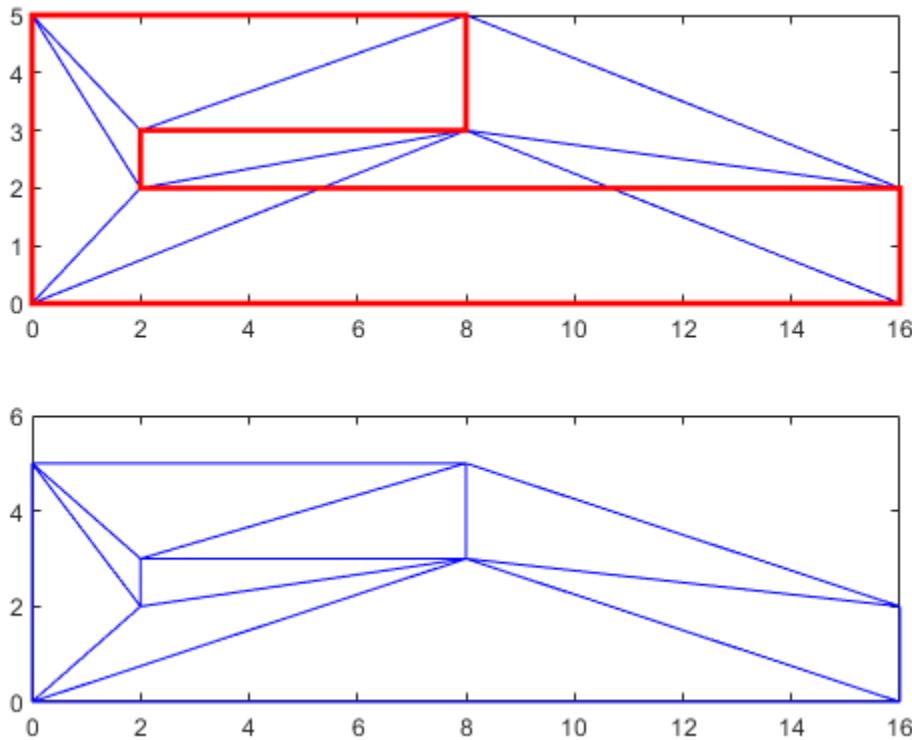


创建并绘制三角剖分以及多边形边界。

```
figure()
subplot(2,1,1);
axis([-1 17 -1 6]);
axis equal

P = [0 0; 16 0; 16 2; 2 2; 2 3; 8 3; 8 5; 0 5];
DT = delaunayTriangulation(P);
triplot(DT)
hold on;
patch(P(:,1),P(:,2),'r','LineWidth',2,'FaceColor',...
    'none','EdgeColor','r');
hold off

% Plot the standalone triangulation in a subplot.
subplot(2,1,2);
axis([-1 17 -1 6]);
axis equal
triplot(DT)
```



此三角剖分不能用于表示多边形域，因为一些三角形横跨了边界。您需要对被三角剖分边切割的边施加约束。由于必须符合所有边的要求，因此需要约束所有边。以下步骤说明如何约束所有边。

输入约束边定义。观察需要约束 ((V1, V2) 间、(V2, V3) 间，以此类推) 的带注释的图窗。

`C = [1 2; 2 3; 3 4; 4 5; 5 6; 6 7; 7 8; 8 1];`

一般而言，如果在定义多边形边界的序列中有 N 个点，则可以将约束表示为 `C = [(1:(N-1))' (2:N)'; N 1];;`

在创建 `delaunayTriangulation` 时指定约束。

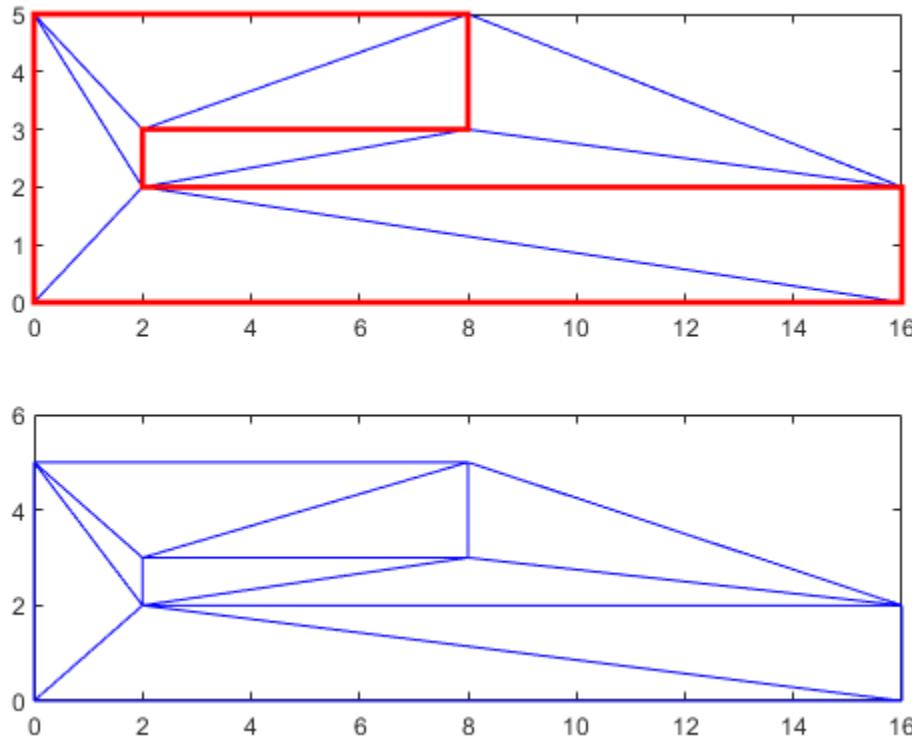
`DT = delaunayTriangulation(P,C);`

也可以通过设置 `Constraints` 属性：`DT.Constraints = C;`，来对现有的三角剖分施加约束。

绘制三角剖分和多边形。

```
figure('Color','white')
subplot(2,1,1);
axis([-1 17 -1 6]);
axis equal
triplot(DT)
hold on;
patch(P(:,1),P(:,2),'r','LineWidth',2, ...
    'FaceColor','none','EdgeColor','r');
hold off
```

```
% Plot the standalone triangulation in a subplot.
subplot(2,1,2);
axis([-1 17 -1 6]);
axis equal
triplot(DT)
```



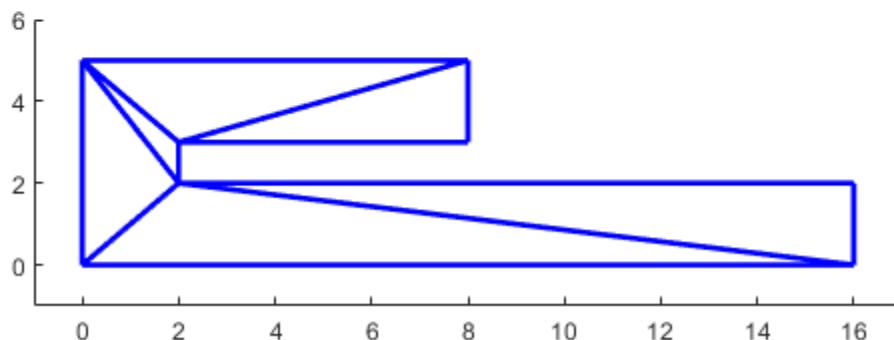
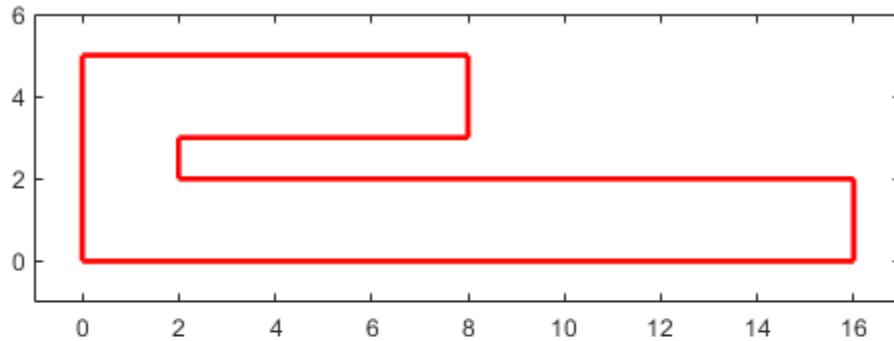
该绘图显示，三角剖分的边符合多边形边界要求。但三角剖分会填充凹处。这种情况下需要一个表示多边形域的三角剖分。您可以使用 `delaunayTriangulation` 方法 `isInterior` 从多边形内提取三角形。此方法会返回一个逻辑数组，其 `true` 值和 `false` 值指示三角形是否在有界的几何域内。该分析基于约当曲线定理，并且通过边约束定义边界。如果第 i 个逻辑标志为 `true`，则三角剖分中的第 i 个三角形被视为在域内，否则被视为在域外。

现在使用 `isInterior` 方法来计算并绘制域中的一系列三角形。

```
% Plot the constrained edges in red.
figure('Color','white')
subplot(2,1,1);
plot(P(C),P(C'+size(P,1)),'-r','LineWidth', 2);
axis([-1 17 -1 6]);

% Compute the in/out status.
IO = isInterior(DT);
subplot(2,1,2);
hold on;
axis([-1 17 -1 6]);
```

```
% Use triplot to plot the triangles that are inside.
% Uses logical indexing and dt(i,j) shorthand
% format to access the triangulation.
triplot(DT(IO,:),DT.Points(:,1), DT.Points(:,2),'LineWidth', 2)
hold off;
```



包含重复位置的点集的三角剖分

MATLAB 中的 Delaunay 算法可以基于特征点集构建三角剖分。如果传递到三角剖分函数或类的点不是特征点，会检测到重复位置并忽略重复点。这样生成的三角剖分不会参照原始输入中的一些点，即重复点。使用 `delaunay` 和 `delaunayn` 函数时，重复情况的存在可能没什么影响。但是，由于 `delaunayTriangulation` 类提供的许多查询以索引为基础，因此必须了解 `delaunayTriangulation` 对特征数据集执行三角剖分和处理。因此，基于唯一点集创建索引是惯例。这些数据通过 `delaunayTriangulation` 的 `Points` 属性维护。

以下示例说明了在处理 `delaunayTriangulation` 时参照在 `Points` 属性内存储的特征数据集的重要性：

```
P = gallery('uniformdata',[25 2],0);
P(18,:) = P(8,:);
P(16,:) = P(6,:);
P(12,:) = P(2,:);
```

```
DT = delaunayTriangulation(P)
```

创建三角剖分时，MATLAB 会发出一条警告。`Points` 属性显示已从数据中删除重复点。

```
DT =
delaunayTriangulation with properties:
    Points: [22x2 double]
    ConnectivityList: [31x3 double]
    Constraints: []
```

例如，如果 Delaunay 三角剖分用于计算凸包，则凸包上的点的索引就是关于特征点集 DT.Points 的索引。因此，使用以下代码计算并绘制凸包：

```
K = DT.convexHull();
plot(DT.Points(:,1),DT.Points(:,2),'r');
hold on
plot(DT.Points(K,1),DT.Points(K,2),'-r');
```

如果将包含重复值的原始数据集与 delaunayTriangulation 提供的索引一起使用，则结果不正确。delaunayTriangulation 处理基于特征数据集 DT.Points 的索引。例如，以下内容会生成错误的绘图，因为 K 的索引是关于 DT.Points 的，而不是关于 P 的。

```
K = DT.convexHull();
plot(P(:,1),P(:,2),'r');
hold on
plot(P(K,1),P(K,2),'-r');
```

通过在创建 delaunayTriangulation 之前删除重复项来创建特征数据集，通常更为方便。这样做将消除可能会出现混淆的情况。如下所示，可以使用 unique 函数完成上述操作：

```
P = gallery('uniformdata',[25 2],0);
P(18,:) = P(8,:);
P(16,:) = P(6,:);
P(12,:) = P(2,:);

[~, I, ~] = unique(P,'first','rows');
I = sort(I);
P = P(I,:);
DT = delaunayTriangulation(P) % The point set is unique
```

另请参阅

详细信息

- “空间搜索”（第 7-51 页）

创建和编辑 Delaunay 三角剖分

此示例说明如何使用 `delaunayTriangulation` 类创建、编辑和查询 Delaunay 三角剖分。Delaunay 三角剖分是科学计算中使用最广泛的三角剖分。与三角剖分关联的属性提供了解决各种几何问题的基础。此外，还说明了如何构造受约束的 Delaunay 三角剖分，同时提供了一个涵盖中轴计算和网格变换的应用程序。

示例一：创建和绘制二维 Delaunay 三角剖分

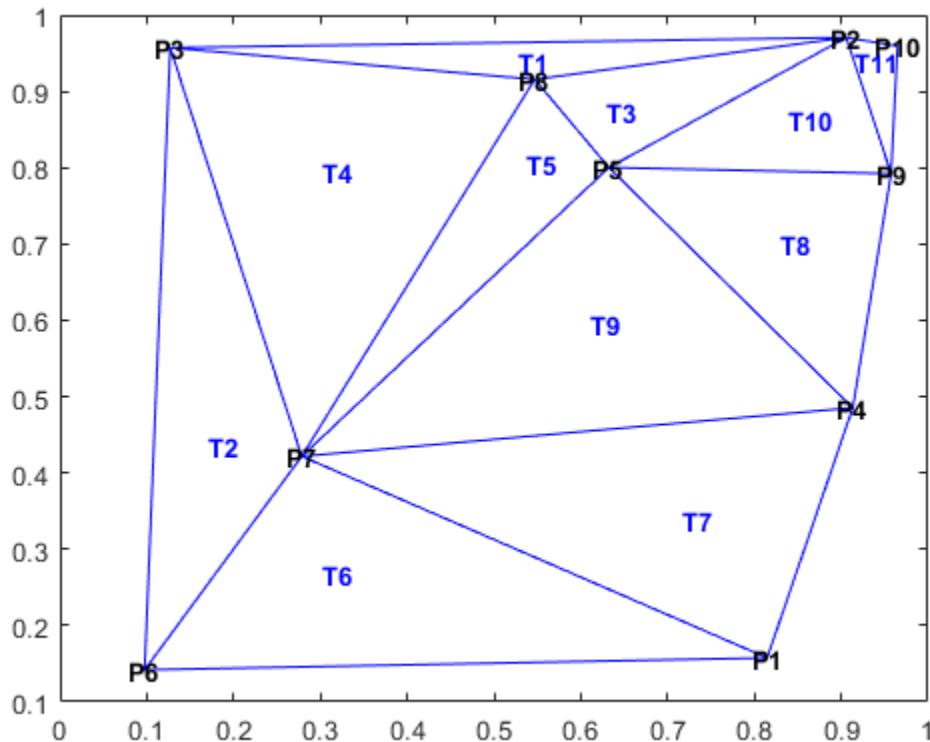
下面的示例说明了如何计算二维 Delaunay 三角剖分，以及如何同时对三角剖分以及顶点和三角标签绘图。

```
x = rand(10,1);
y = rand(10,1);
dt = delaunayTriangulation(x,y)

dt =
delaunayTriangulation with properties:

    Points: [10x2 double]
    ConnectivityList: [11x3 double]
    Constraints: []

triplot(dt);
%
% Display the Vertex and Triangle labels on the plot
hold on
vxlabels = arrayfun(@(n) {sprintf('P%d', n)}, (1:10)');
Hpl = text(x, y, vxlabels, 'FontWeight', 'bold', 'HorizontalAlignment',...
    'center', 'BackgroundColor', 'none');
ic = incenter(dt);
numtri = size(dt,1);
trilabels = arrayfun(@(x) {sprintf('T%d', x)}, (1:numtri)');
Htl = text(ic(:,1), ic(:,2), trilabels, 'FontWeight', 'bold',...
    'HorizontalAlignment', 'center', 'Color', 'blue');
hold off
```



示例二：创建和绘制三维 Delaunay 三角剖分

下面的示例说明了如何计算三维 Delaunay 三角剖分以及如何对三角剖分绘图。

```
X = rand(10,3)
```

```
X = 10×3
```

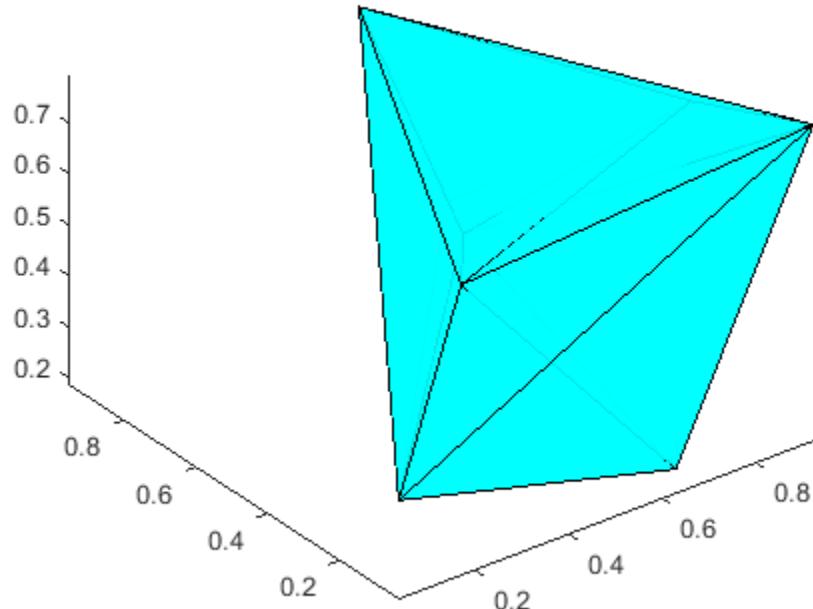
```
0.6557 0.7060 0.4387
0.0357 0.0318 0.3816
0.8491 0.2769 0.7655
0.9340 0.0462 0.7952
0.6787 0.0971 0.1869
0.7577 0.8235 0.4898
0.7431 0.6948 0.4456
0.3922 0.3171 0.6463
0.6555 0.9502 0.7094
0.1712 0.0344 0.7547
```

```
dt = delaunayTriangulation(X)
```

```
dt =
delaunayTriangulation with properties:
```

```
Points: [10x3 double]
ConnectivityList: [22x4 double]
Constraints: []
```

```
tetramesh(dt, 'FaceColor', 'cyan');
```



```
% To display large tetrahedral meshes use the convexHull method to
% compute the boundary triangulation and plot it using trisurf.
% For example;
% triboundary = convexHull(dt)
% trisurf(triboundary, X(:,1), X(:,2), X(:,3), 'FaceColor', 'cyan')
```

示例三：访问三角剖分数据结构体

可通过两种方式访问三角剖分数据结构。一种方法是通过三角剖分属性，另一种方法是使用索引。

通过 10 个随机点创建二维 Delaunay 三角剖分。

```
X = rand(10,2)
```

```
X = 10×2
```

0.2760	0.7513
0.6797	0.2551
0.6551	0.5060
0.1626	0.6991
0.1190	0.8909
0.4984	0.9593
0.9597	0.5472
0.3404	0.1386
0.5853	0.1493
0.2238	0.2575

```

dt = delaunayTriangulation(X)

dt =
delaunayTriangulation with properties:

    Points: [10x2 double]
    ConnectivityList: [12x3 double]
    Constraints: []

% The triangulation datastructure is;
dt.ConnectivityList

ans = 12×3

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

% Indexing is a shorthand way to query the triangulation. The format is
% dt(i, j) where j is the j'th vertex of the i'th triangle, standard
% indexing rules apply.
% The triangulation datastructure is
dt(:,i)

ans = 12×3

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

第二个三角形为:

dt(2,:)

ans = 1×3

3   8   9

第二个三角形的第三个顶点为:

```

```
dt(2,3)
```

```
ans = 9
```

前三个三角形为：

```
dt(1:3,:)
```

```
ans = 3×3
```

4	10	1
3	8	9
10	4	5

示例四：编辑 Delaunay 三角剖分以插入或删除点

下面的示例说明了如何使用基于索引的下标插入或删除点。相对于从头重新创建新的 delaunayTriangulation，编辑 delaunayTriangulation 以进行少量修改的效率更高，数据集较大时尤其如此。

```
% Construct a Delaunay triangulation from
% 10 random points within a unit square
x = rand(10,1);
y = rand(10,1);
dt = delaunayTriangulation(x,y)
```

```
dt =
delaunayTriangulation with properties:
```

```
Points: [10x2 double]
ConnectivityList: [12x3 double]
Constraints: []
```

```
% Insert 5 additional random points
dt.Points(end+(1:5),:) = rand(5,2)
```

```
dt =
delaunayTriangulation with properties:
```

```
Points: [15x2 double]
ConnectivityList: [21x3 double]
Constraints: []
```

替换第五个点

```
dt.Points(5,:) = [0, 0]
```

```
dt =
delaunayTriangulation with properties:
```

```
Points: [15x2 double]
ConnectivityList: [21x3 double]
Constraints: []
```

删除第四个点

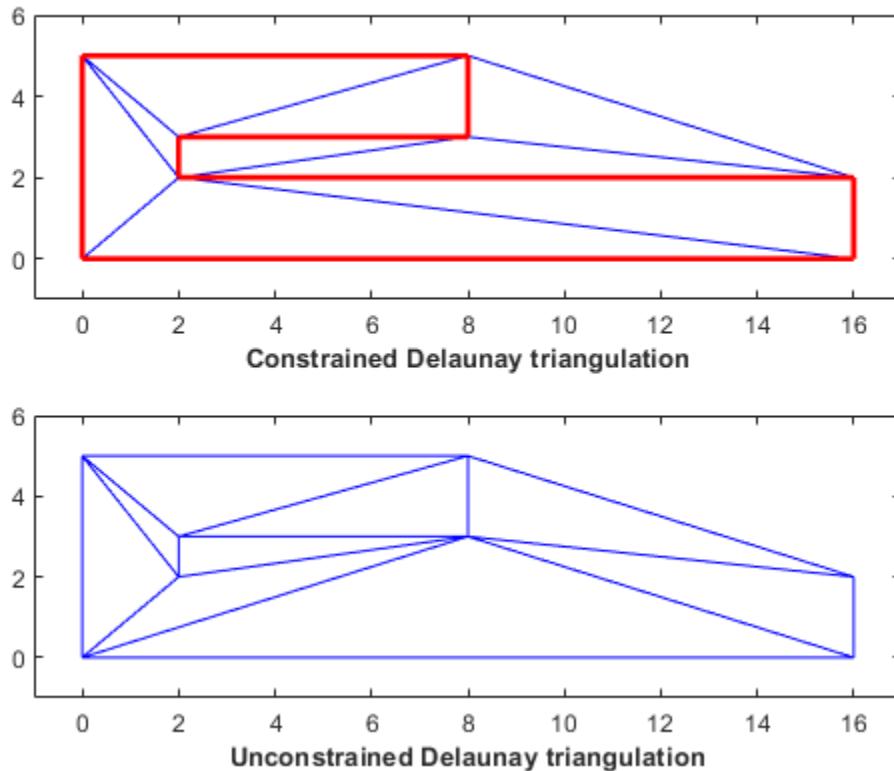
```
dt.Points(4,:) = []
dt =
delaunayTriangulation with properties:
    Points: [14x2 double]
    ConnectivityList: [19x3 double]
    Constraints: []
```

示例五：创建受约束的 Delaunay 三角剖分

下面的示例说明了如何创建一个简单的受约束的 Delaunay 三角剖分，并描绘了约束的影响。

```
X = [0 0; 16 0; 16 2; 2 2; 2 3; 8 3; 8 5; 0 5];
C = [1 2; 2 3; 3 4; 4 5; 5 6; 6 7; 7 8; 8 1];
dt = delaunayTriangulation(X, C);
subplot(2,1,1);
triplot(dt);
axis([-1 17 -1 6]);
xlabel('Constrained Delaunay triangulation', 'fontWeight','b');
% Plot the constrained edges in red
hold on;
plot(X(C),X(C+size(X,1)),'r', 'LineWidth', 2);
hold off;

% Now delete the constraints and plot the unconstrained Delaunay
dt.Constraints = [];
subplot(2,1,2);
triplot(dt);
axis([-1 17 -1 6]);
xlabel('Unconstrained Delaunay triangulation', 'fontWeight','b');
```



示例六：创建地图的受约束的 Delaunay 三角剖分

加载美国本土的周长地图。构造一个受约束的代表该多边形的 Delaunay 三角剖分。该三角剖分跨受点集的凸包约束的一个域。筛选出多边形域内的三角形并对其绘图。注意：该数据集包含重复的数据点；即两个或更多个数据点的位置相同。重复的点将被拒绝，delaunayTriangulation 会相应地重新设置约束的格式。

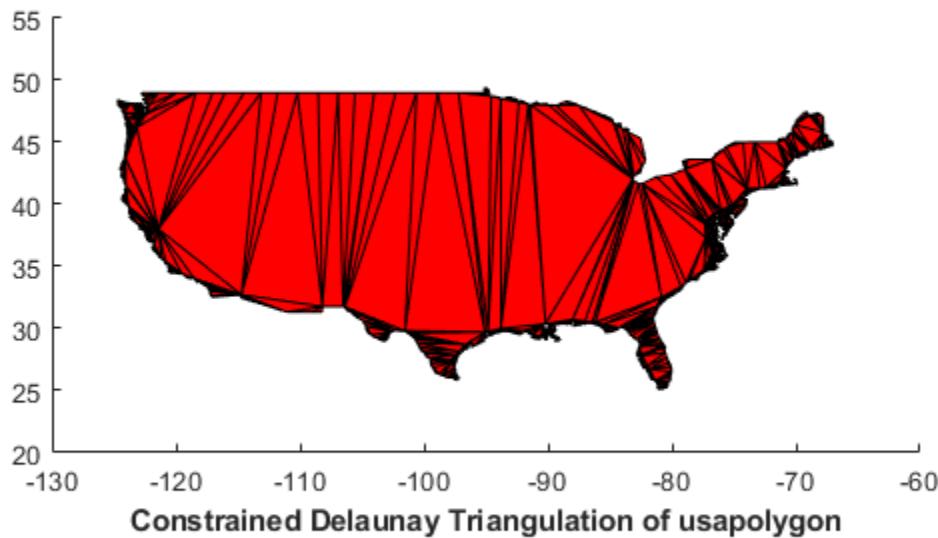
```
clf
load usapolygon
% Define an edge constraint between two successive
% points that make up the polygonal boundary.
nump = numel(uslon);
C = [(1:(nump-1))' (2:nump)'; nump 1];
dt = delaunayTriangulation(uslon, uslat, C);
```

Warning: Duplicate data points have been detected and removed.

The Triangulation indices and constraints are defined with respect to the unique set of points in delaunayTriangulation.

Warning: Intersecting edge constraints have been split, this may have added new points into the triangulation.

```
io = dt.isInterior();
patch('faces',dt(io,:),'vertices', dt.Points, 'FaceColor','r');
axis equal;
axis([-130 -60 20 55]);
xlabel('Constrained Delaunay Triangulation of usapolygon', 'fontweight','b');
```



示例七：从点云重新构造曲线

下面的示例重点介绍如何使用 Delaunay 三角剖分从点云重新构造多边形边界。重新构造操作基于精心设计的 Crust 算法。

参考资料：N. Amenta、M. Bern 和 D. Eppstein。The crust and the beta-skeleton: combinatorial curve reconstruction. Graphical Models and Image Processing, 60:125-135, 1998.

```
% Create a set of points representing the point cloud
numpts = 192;
t = linspace( -pi, pi, numpts+1 )';
t(end) = [];
r = 0.1 + 5*sqrt( cos( 6*t ).^2 + (0.7).^2 );
x = r.*cos(t);
y = r.*sin(t);
ri = randperm(numpts);
x = x(ri);
y = y(ri);

% Construct a Delaunay Triangulation of the point set.
dt = delaunayTriangulation(x,y);
tri = dt(:,__);

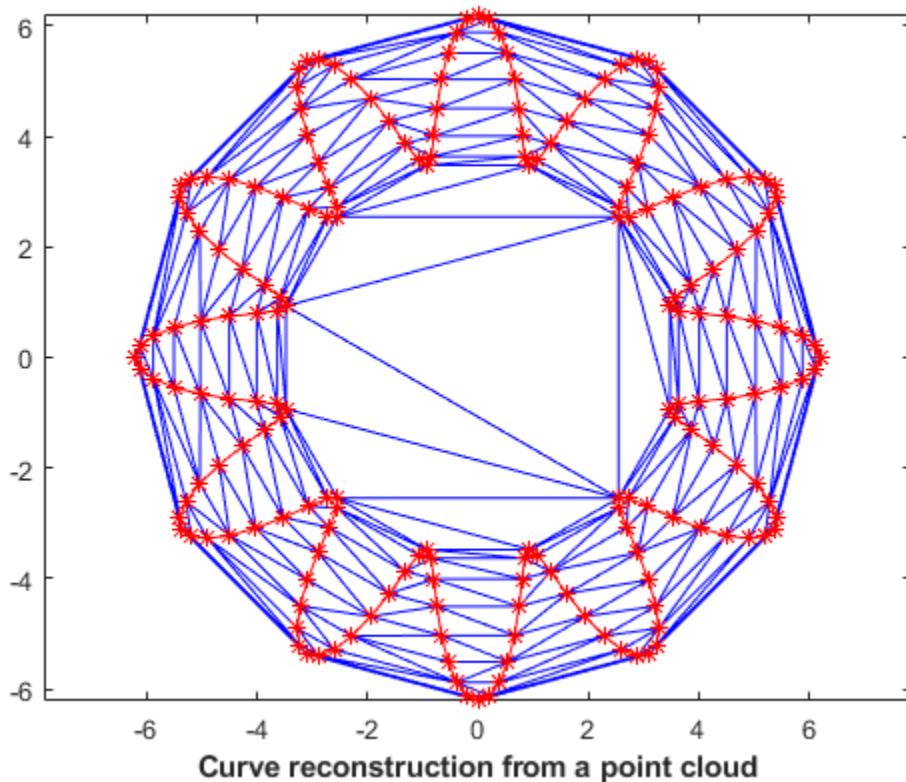
% Insert the location of the Voronoi vertices into the existing
% triangulation
V = dt.voronoiDiagram();
% Remove the infinite vertex
V(1,:) = [];
```

```
numv = size(V,1);
dt.Points(end+(1:numv),:) = V;
```

Warning: Duplicate data points have been detected and removed.

The Triangulation indices are defined with respect to the unique set of points in delaunayTriangulation.

```
% The Delaunay edges that connect pairs of sample points represent the
% boundary.
delEdges = dt.edges();
validx = delEdges(:,1) <= numpts;
validy = delEdges(:,2) <= numpts;
boundaryEdges = delEdges((validx & validy), :)';
xb = x(boundaryEdges);
yb = y(boundaryEdges);
clf;
triplot(tri,x,y);
axis equal;
hold on;
plot(x,y,'*r');
plot(xb,yb, '-r');
xlabel('Curve reconstruction from a point cloud', 'fontweight','b');
hold off;
```



示例八：计算多边形域的近似中轴

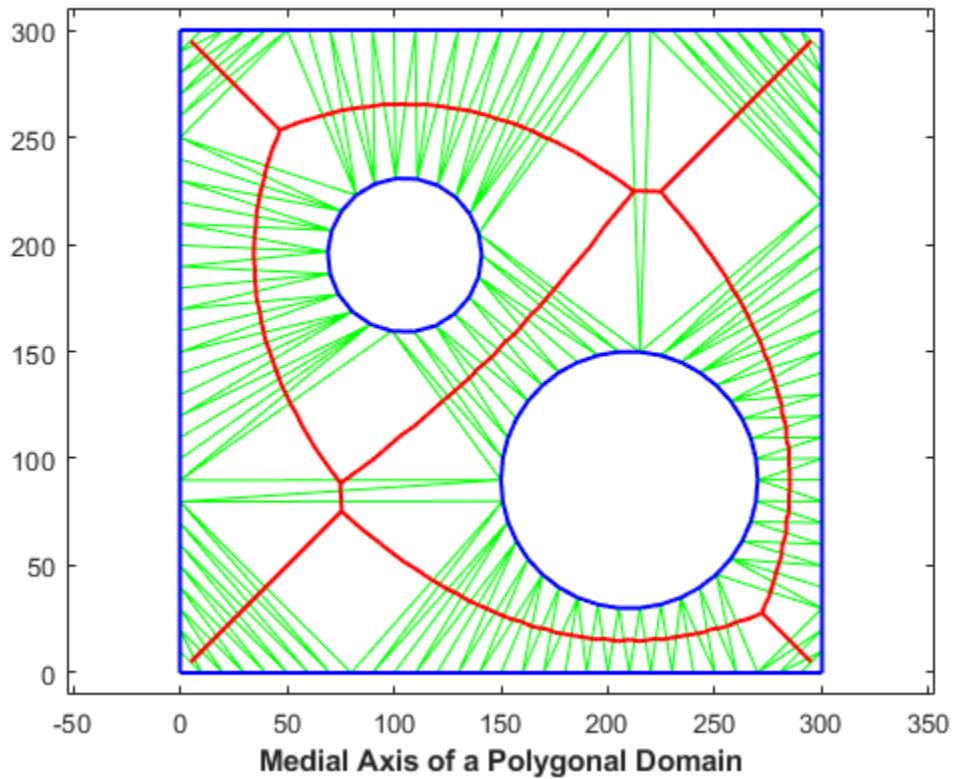
此示例说明如何使用受约束的 Delaunay 三角剖分创建一个多边形域的近似中轴。多边形的中轴由多边形内部最大圆的中心的轨迹定义。

```
% Construct a constrained Delaunay triangulation of a sample of points
% on the domain boundary.
load trimesh2d
dt = delaunayTriangulation(x,y,Constraints);
inside = dt.isInterior();

% Construct a triangulation to represent the domain triangles.
tr = triangulation(dt(inside, :), dt.Points);

% Construct a set of edges that join the circumcenters of neighboring
% triangles; the additional logic constructs a unique set of such edges.
numt = size(tr,1);
T = (1:numt)';
neigh = tr.neighbors();
cc = tr.circumcenter();
xcc = cc(:,1);
ycc = cc(:,2);
idx1 = T < neigh(:,1);
idx2 = T < neigh(:,2);
idx3 = T < neigh(:,3);
neigh = [T(idx1) neigh(idx1,1); T(idx2) neigh(idx2,2); T(idx3) neigh(idx3,3)]';

% Plot the domain triangles in green, the domain boundary in blue and the
% medial axis in red.
clf;
triplot(tr, 'g');
hold on;
plot(xcc(neigh), ycc(neigh), '-r', 'LineWidth', 1.5);
axis([-10 310 -10 310]);
axis equal;
plot(x(Constraints'),y(Constraints'), '-b', 'LineWidth', 1.5);
xlabel('Medial Axis of a Polygonal Domain', 'fontWeight','bold');
hold off;
```

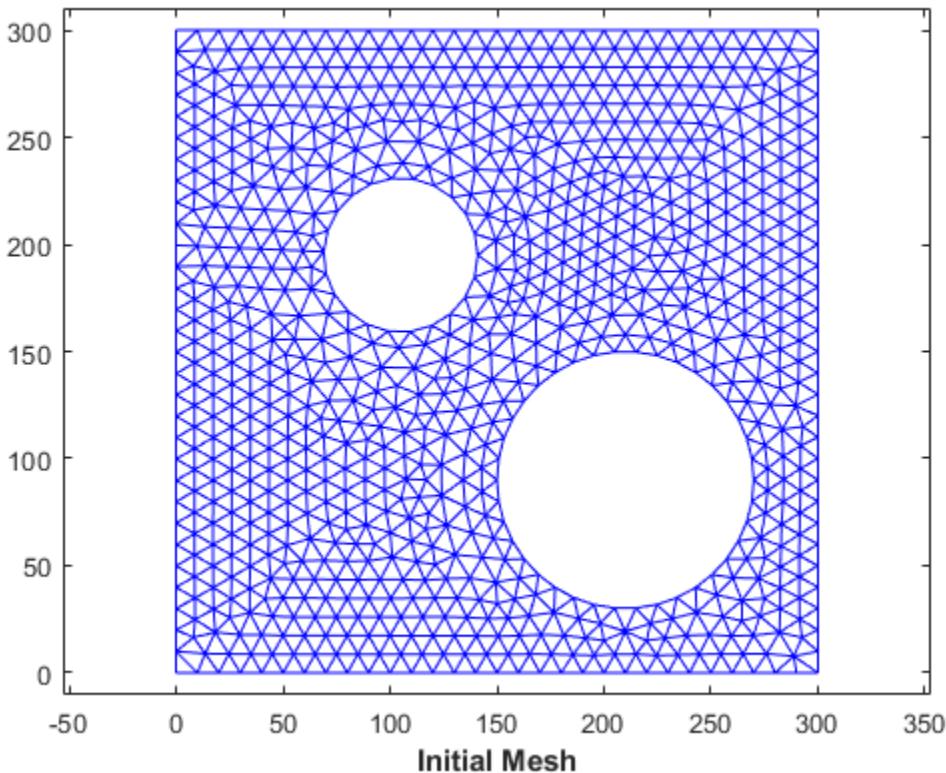


示例九：按照修改的边界变换二维网格

下面的示例说明了如何变换二维域的网格以适应对域边界的修改。

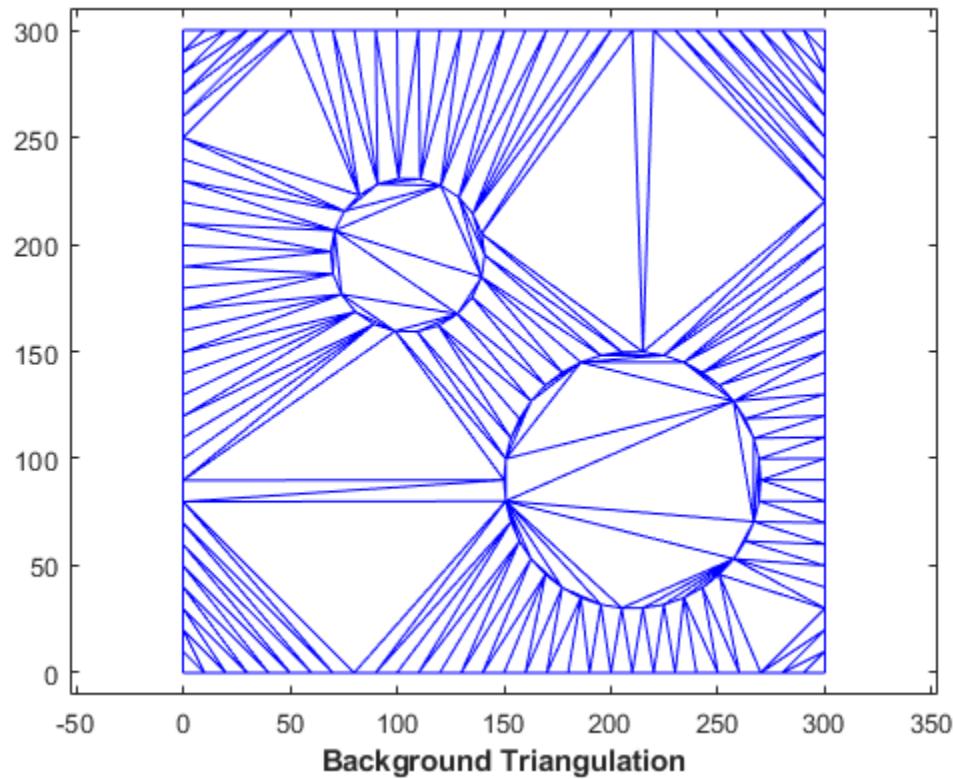
步骤 1：加载数据。要变换的网格由 trife、xfe、yfe 定义，它是面-顶点格式的三角剖分。

```
load trimesh2d
clf;
triplot(trife,xfe,yfe);
axis equal;
axis([-10 310 -10 310]);
axis equal;
xlabel('Initial Mesh', 'fontweight','b');
```



步骤 2：构造背景三角剖分 - 由代表网格边界的点集构成的受约束的 Delaunay 三角剖分。对于网格的每个顶点，计算用于定义与其背景三角剖分相关的位置的描述符。该描述符表示该封闭三角形以及与该三角形相关的重心坐标。

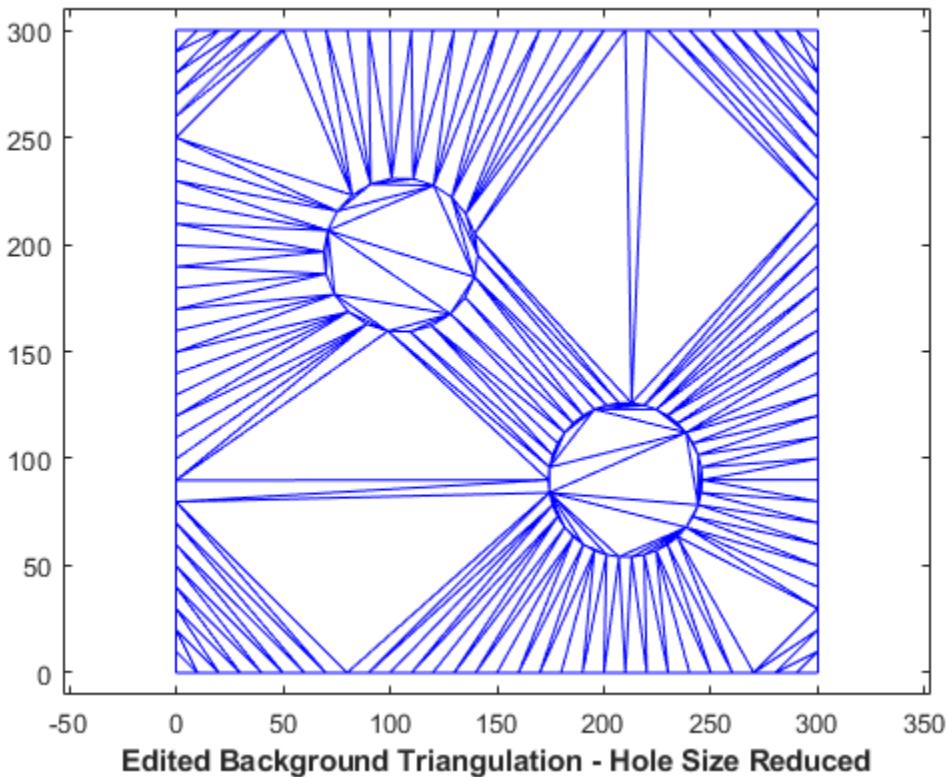
```
dt = delaunayTriangulation(x,y,Constraints);
clf;
triplot(dt);
axis equal;
axis([-10 310 -10 310]);
axis equal;
xlabel('Background Triangulation', 'fontweight','b');
```



```
descriptors.tri = pointLocation(dt,xfe, yfe);
descriptors.baryCoords = cartesianToBarycentric(dt,descriptors.tri, [xfe yfe]);
```

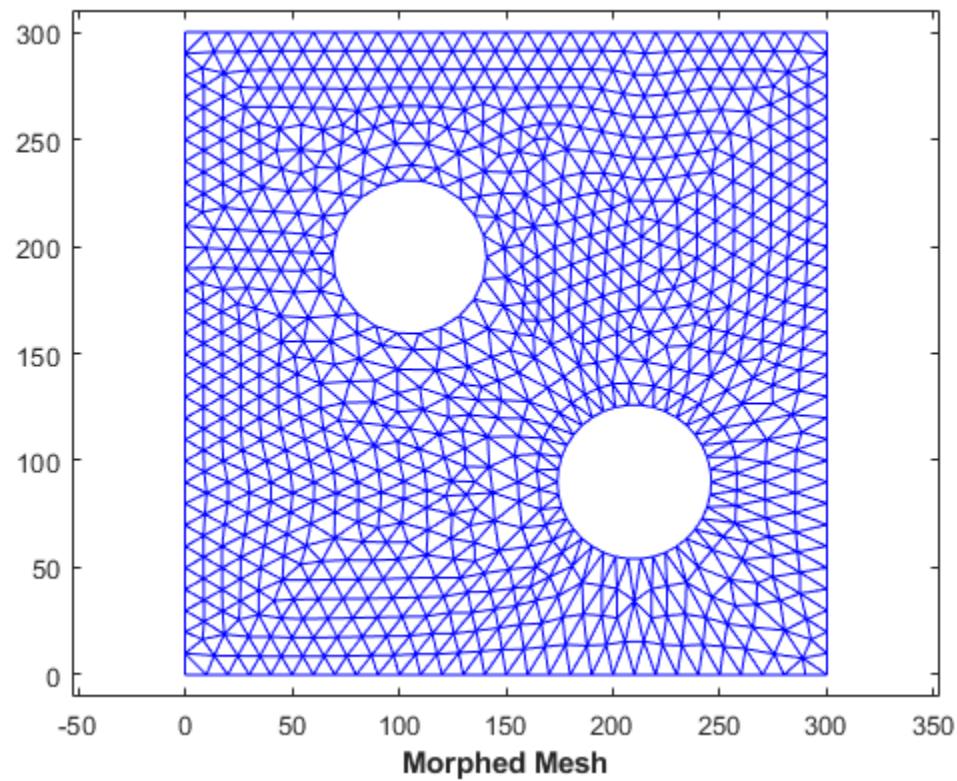
步骤 3：编辑背景三角剖分以将需要的修改融入域边界中。

```
cc1 = [210 90];
circ1 = (143:180)';
x(circ1) = (x(circ1)-cc1(1))*0.6 + cc1(1);
y(circ1) = (y(circ1)-cc1(2))*0.6 + cc1(2);
tr = triangulation(dt(:,:,x,y));
clf;
triplot(tr);
axis([-10 310 -10 310]);
axis equal;
xlabel('Edited Background Triangulation - Hole Size Reduced', 'fontweight','b');
```



步骤 4：使用变形的背景三角剖分作为计算的基础，将该描述符重新转换为笛卡尔坐标。

```
Xnew = barycentricToCartesian(tr,descriptors.tri, descriptors.baryCoords);
tr = triangulation(trife, Xnew);
clf;
triplot(tr);
axis([-10 310 -10 310]);
axis equal;
xlabel('Morphed Mesh', 'fontweight','b');
```



空间搜索

本节内容

- “简介” (第 7-51 页)
- “最近邻点搜索” (第 7-51 页)
- “点位置搜索” (第 7-53 页)

简介

MATLAB 提供了使用 Delaunay 三角剖分或常规三角剖分执行空间搜索所必需的函数。MATLAB 支持的搜索查询包括：

- 最近邻点搜索（有时称为最近点搜索或邻近搜索）。
- 点位置搜索（有时称为三角形内的点搜索或单纯形内的点搜索，其中单纯形是三角形、四面体或更高维度的等效形状）。

如果是欧几里德空间中的点集 X 和查询点 q ，最近邻点搜索将查找 X 中比 X 中的任何其他点都邻近 q 的点 p 。如果是 X 的三角剖分，点位置搜索将查找包含查询点 q 的三角形或四面体。由于这些方法适用于 Delaunay 三角剖分以及常规三角剖分，因此即使对点的修改违反了 Delaunay 准则，也可以使用它们。您也可以搜索表示为矩阵格式的常规三角剖分。

尽管 MATLAB 支持在 N 维中执行这些搜索方案，但维数在三维以上时通常禁止使用精确的空间搜索。应考虑使用近似的替代方法来解决多达 10 维时的大型问题。

最近邻点搜索

根据问题的维度，MATLAB 中有多种方法可计算最近邻点：

- 对于二维和三维搜索，使用 `triangulation` 类提供并由 `delaunayTriangulation` 类继承的 `nearestNeighbor` 方法。
- 对于 4 维和更高维，使用 `delaunayn` 函数构建三角剖分，并使用补充的 `dsearchn` 函数执行搜索。虽然这些 N 维函数支持二维和三维，但它们并没有三角剖分提供的搜索方法通用和高效。

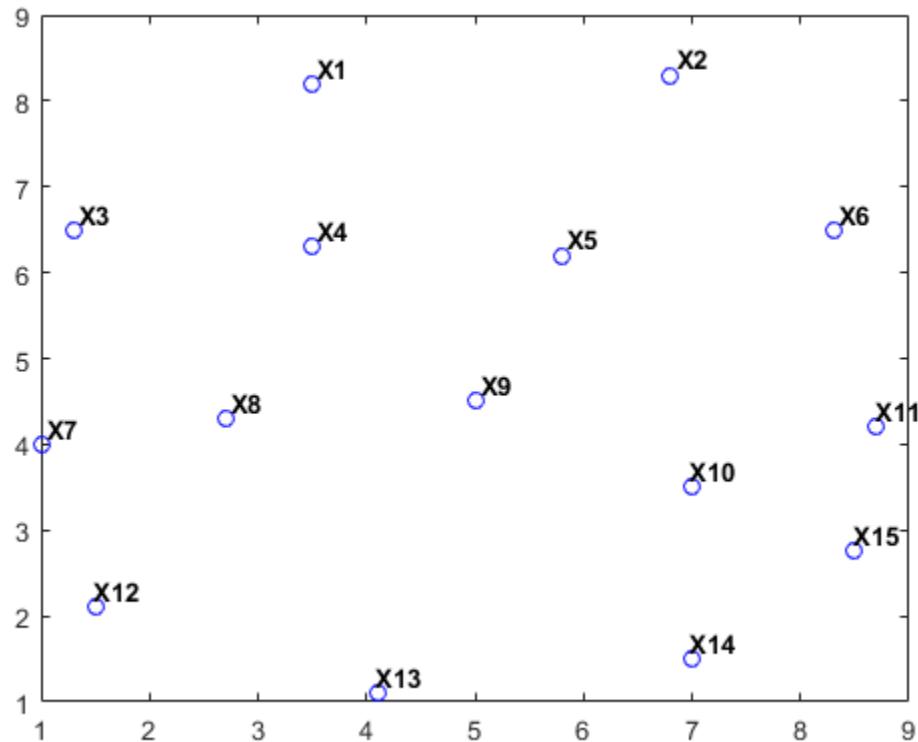
此示例说明如何使用 `delaunayTriangulation` 执行二维最近邻搜索。

首先创建一个包含 15 个点的随机点集。

```
X = [3.5 8.2; 6.8 8.3; 1.3 6.5; 3.5 6.3; 5.8 6.2; 8.3 6.5;...
    1 4; 2.7 4.3; 5 4.5; 7 3.5; 8.7 4.2; 1.5 2.1; 4.1 1.1; ...
    7 1.5; 8.5 2.75];
```

绘制这些点并添加注释，以显示 ID 标签。

```
plot(X(:,1),X(:,2),'ob')
hold on
vxlabels = arrayfun(@(n) {sprintf('X%d', n)}, (1:15));
Hpl = text(X(:,1)+0.2, X(:,2)+0.2, vxlabels, 'FontWeight',...
    'bold', 'HorizontalAlignment','center', 'BackgroundColor',...
    'none');
hold off
```



创建这些点的 Delaunay 三角剖分。

```
dt = delaunayTriangulation(X);
```

创建一些查询点，并使用 `nearestNeighbor` 方法针对每个查询点在 `X` 中查找其对应的最近邻的索引。

```
numq = 10;
rng(0,'twister');
q = 2+rand(numq,2)*6;
xi = nearestNeighbor(dt, q);
```

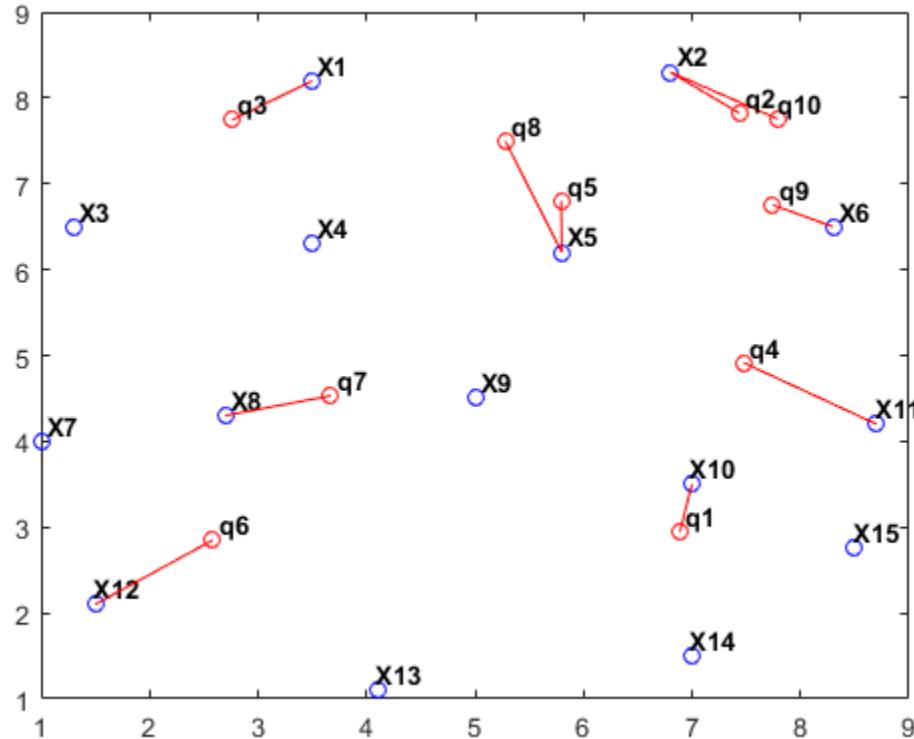
在绘图中添加查询点，并添加将查询点连接到其最近邻的线段。

```
xnn = X(xi,:);

hold on
plot(q(:,1),q(:,2),'or');
plot([xnn(:,1) q(:,1)]',[xnn(:,2) q(:,2)]','-r');

vxlabels = arrayfun(@(n) {sprintf('q%d', n)}, (1:numq));
Hpl = text(q(:,1)+0.2, q(:,2)+0.2, vxlabels, 'FontWeight',...
    'bold', 'HorizontalAlignment','center',...
    'BackgroundColor','none');

hold off
```



执行三维最近邻搜索是对基于 `delaunayTriangulation` 的二维示例的直接扩展。

对于四维以及更高维而言，使用以下示例中所述的 `delaunayn` 和 `dsearchn` 函数：

创建四维随机样本点，并使用 `delaunayn` 对这些点执行三角剖分：

```
X = 20*rand(50,4) -10;
tri = delaunayn(X);
```

创建一些查询点，使用 `dsearchn` 函数针对每个查询点在 `X` 间查找与其对应的最近邻点的索引：

```
q = rand(5,4);
xi = dsearchn(X,tri, q)
```

`nearestNeighbor` 方法和 `dsearchn` 函数允许以可选参数的形式返回查询点与其最近邻点之间的欧几里得距离。在四维示例中，可以计算距离 (`dnn`)，如下所示：

```
[xi,dnn] = dsearchn(X,tri, q)
```

点位置搜索

点位置搜索是一种通过封闭查询点来确定单纯形（三角形、四面体等）的三角剖分搜索算法。在最近邻点搜索的情况下，根据问题的维度，在 MATLAB 中执行点位置搜索有多种方法：

- 对二维和三维，使用基于类的方式，以及由 `triangulation` 类提供并由 `delaunayTriangulation` 类继承的 `pointLocation` 方法。

- 对于 4 维和更高维，使用 **delaunayn** 函数构建三角剖分，并使用补充的 **tsearchn** 函数执行点位置搜索。虽然这些 N 维函数支持二维和三维，但它们并没有三角剖分提供的搜索方法通用和高效。

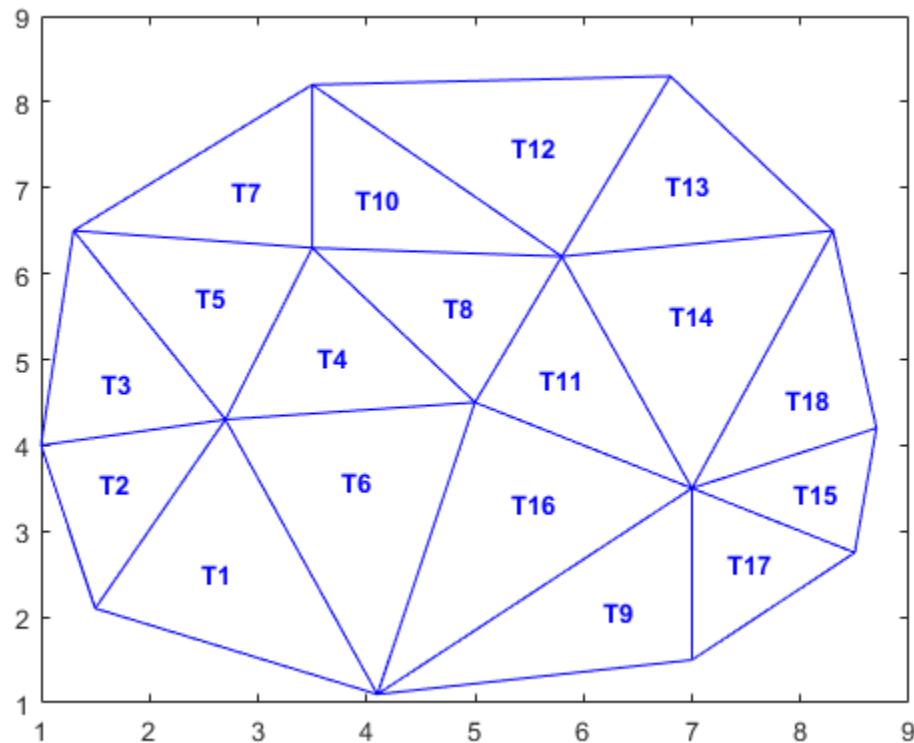
此示例说明如何使用 **delaunayTriangulation** 类来执行二维点位置搜索。

首先从一个二维点集开始。

```
X = [3.5 8.2; 6.8 8.3; 1.3 6.5; 3.5 6.3; 5.8 6.2; ...
      8.3 6.5; 1 4; 2.7 4.3; 5 4.5; 7 3.5; 8.7 4.2; ...
      1.5 2.1; 4.1 1.1; 7 1.5; 8.5 2.75];
```

创建并绘制三角剖分，在三角形的内心显示三角形 ID 标签。

```
dt = delaunayTriangulation(X);
trilabels = arrayfun(@(x) {sprintf('T%d', x)}, (1:size(dt,1)));
Htl = text(ic(:,1), ic(:,2), trilabels, 'FontWeight', ...
           'bold', 'HorizontalAlignment', 'center', 'Color', ...
           'blue');
hold on
ic = incenter(dt);
numtri = size(dt,1);
trilabels = arrayfun(@(x) {sprintf('T%d', x)}, (1:numtri));
Htl = text(ic(:,1), ic(:,2), trilabels, 'FontWeight', ...
           'bold', 'HorizontalAlignment', 'center', 'Color', ...
           'blue');
hold off
```



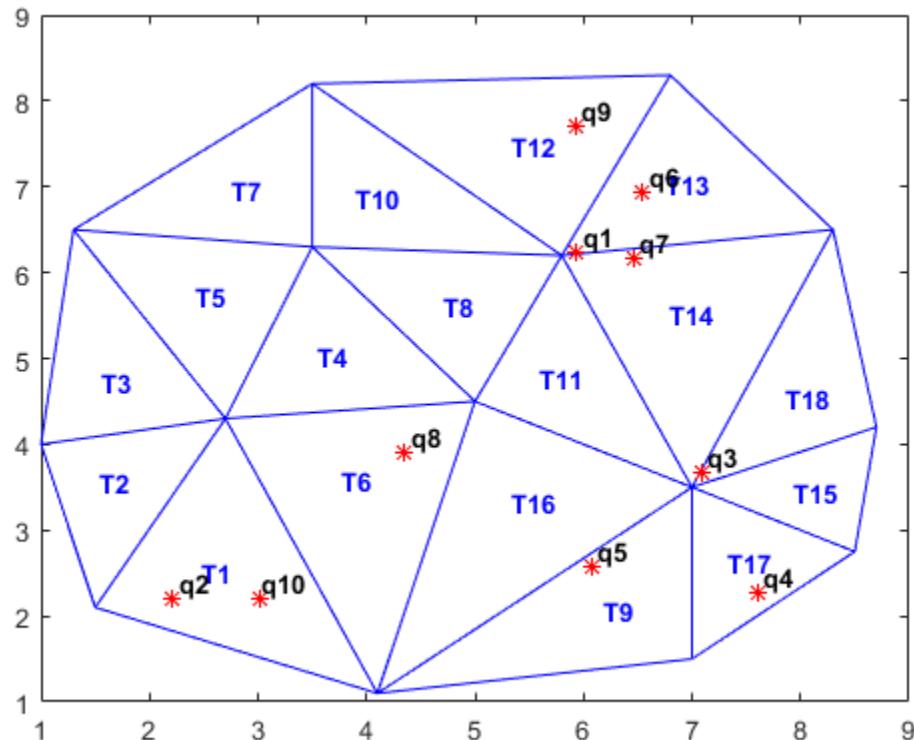
现在创建一些查询点，并将其添加到绘图中。然后使用 **pointLocation** 方法查找对应的包围三角形的索引。

```

q = [5.9344  6.2363;
      2.2143  2.1910;
      7.0948  3.6615;
      7.6040  2.2770;
      6.0724  2.5828;
      6.5464  6.9407;
      6.4588  6.1690;
      4.3534  3.9026;
      5.9329  7.7013;
      3.0271  2.2067];

hold on;
plot(q(:,1),q(:,2),'*r');
vxlabels = arrayfun(@(n) {sprintf('q%d', n)}, (1:10)');
Hpl = text(q(:,1)+0.2, q(:,2)+0.2, vxlabels, 'FontWeight',...
    'bold', 'HorizontalAlignment','center',...
    'BackgroundColor', 'none');
hold off

```



ti = pointLocation(dt,q);

执行三维点位置搜索是对使用 `delaunayTriangulation` 执行二维点位置搜索的直接扩展。

对于四维以及更高维而言，使用以下示例中所述的 `delaunayn` 和 `tsearchn` 函数：

创建四维随机样本点，然后使用 `delaunayn` 进行三角剖分：

```
X = 20*rand(50,4) -10;
tri = delaunayn(X);
```

使用 `tsearchn` 函数创建一些查询点，求出对应封闭单纯形的索引：

```
q = rand(5,4);
ti = tsearchn(X,tri,q)
```

`pointLocation` 方法和 `tsearchn` 函数允许以可选参数的形式返回对应的重心坐标。在四维示例中，可按如下方式计算重心坐标：

```
[ti,bc] = tsearchn(X,tri,q)
```

重心坐标对执行线性插值很有用。这些坐标提供了可用于在封闭单纯形的每个顶点缩放数值的权值。有关详细信息，请参阅“内插散点数据”（第 8-37 页）。

另请参阅

`delaunay` | `delaunayTriangulation` | `delaunayn` | `dsearchn` | `nearestNeighbor` |
`pointLocation` | `triangulation` | `triangulation` | `tsearchn`

相关示例

- “使用 Delaunay 三角剖分”（第 7-13 页）
- “三角剖分表示法”（第 7-2 页）
- “内插散点数据”（第 8-37 页）

Voronoi 图

本节内容

“绘图二维 Voronoi 图和 Delaunay 三角剖分”（第 7-57 页）

“计算 Voronoi 图”（第 7-60 页）

离散点集 X 的 Voronoi 图将每个点 $X(i)$ 周围的空间分解成影响区域 $R\{i\}$ 。这种分解具有的属性是区域 $R\{i\}$ 中的任意点 P 比任何其他点更靠近点 i 。这种影响区域称为 Voronoi 区域，所有 Voronoi 区域的集合构成 Voronoi 图。

Voronoi 图是一个 N 维几何结构，但是大多数实际应用程序是位于二维和三维空间中的。使用一个示例最好理解 Voronoi 图的属性。

绘图二维 Voronoi 图和 Delaunay 三角剖分

本示例在同一个二维图上显示 Voronoi 图和 Delaunay 三角剖分。

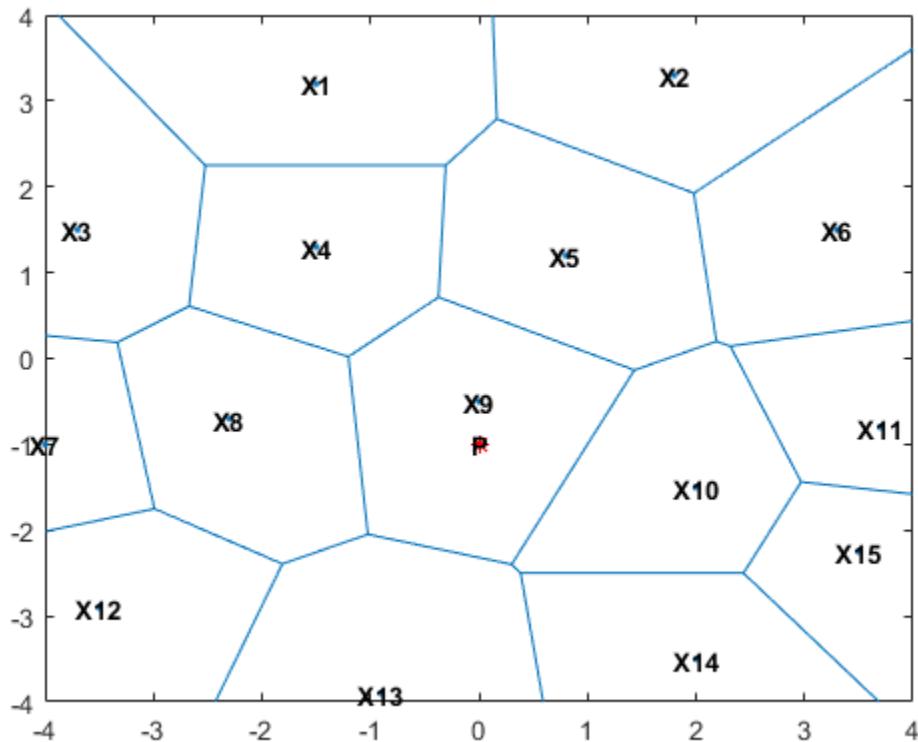
使用二维 `voronoi` 函数绘制某一点集的 Voronoi 图。

```
figure()
X = [-1.5 3.2; 1.8 3.3; -3.7 1.5; -1.5 1.3; ...
      0.8 1.2; 3.3 1.5; -4.0 -1.0; -2.3 -0.7; ...
      0 -0.5; 2.0 -1.5; 3.7 -0.8; -3.5 -2.9; ...
      -0.9 -3.9; 2.0 -3.5; 3.5 -2.25];

voronoi(X(:,1),X(:,2))

% Assign labels to the points.
nump = size(X,1);
plabels = arrayfun(@(n) {sprintf('X%d', n)}, (1:nump)');
hold on
Hpl = text(X(:,1), X(:,2), plabels, 'FontWeight', ...
            'bold', 'HorizontalAlignment','center',...
            'BackgroundColor', 'none');

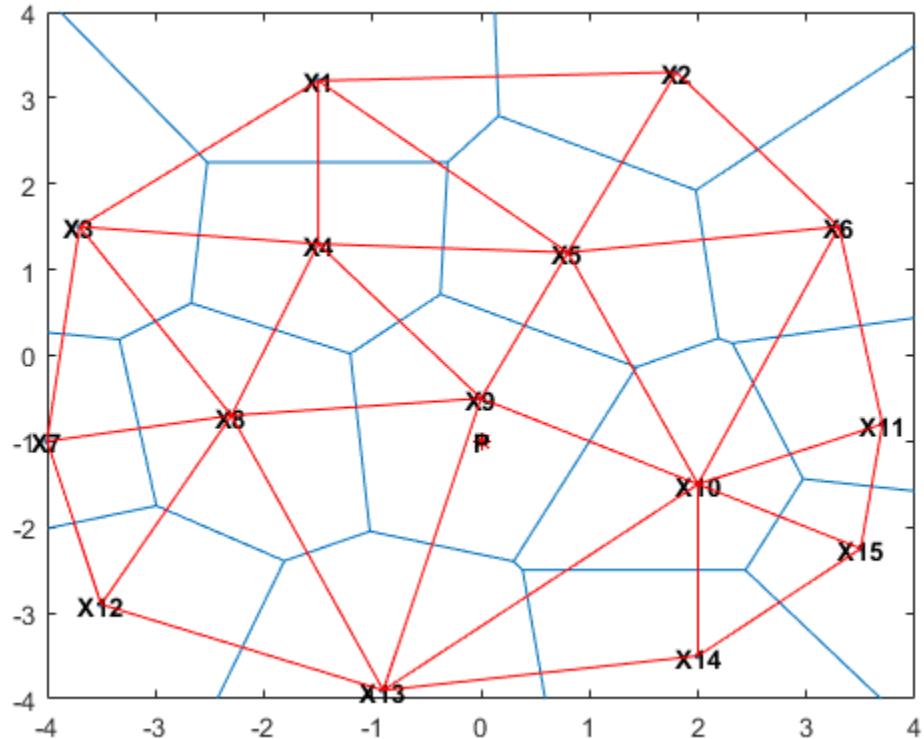
% Add a query point, P, at (0, -1.5).
P = [0 -1];
plot(P(1),P(2), '*r');
text(P(1), P(2), 'P', 'FontWeight', 'bold',...
            'HorizontalAlignment','center',...
            'BackgroundColor', 'none');
hold off
```



观察发现 P 比 X 中的任何其他点更靠近 X_9 ，对于约束 X_9 的区域中的任何点 P 都是如此。

点集 X 的 Voronoi 图与 X 的 Delaunay 三角剖分密切相关。要了解这种关系，请构建点集 X 的一个 Delaunay 三角剖分，然后在 Voronoi 图上叠加该三角剖分图。

```
dt = delaunayTriangulation(X);
hold on
triplot(dt,'r');
hold off
```



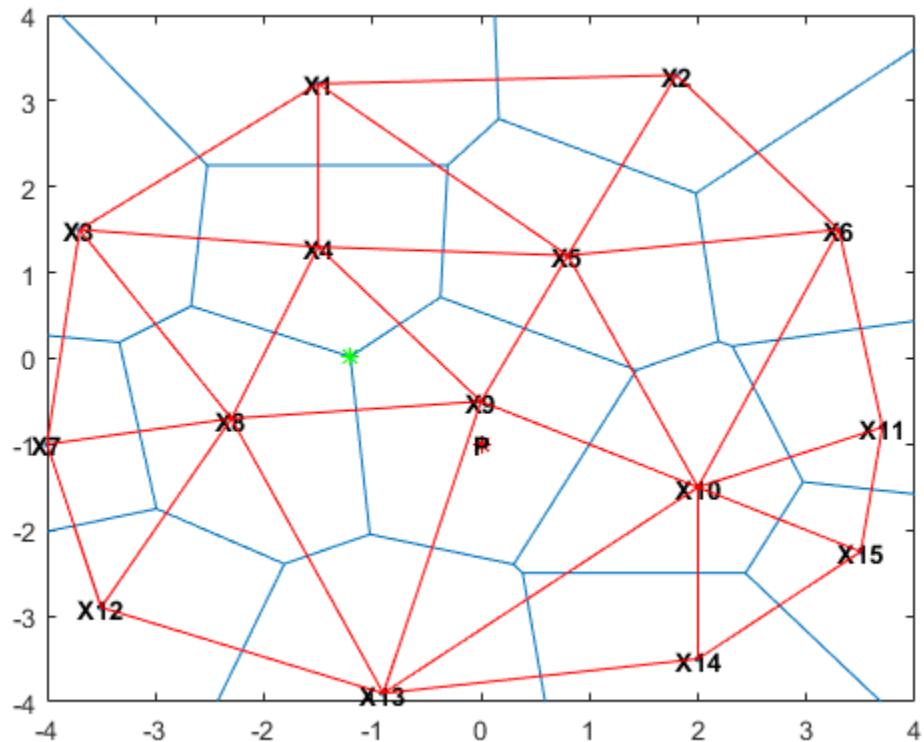
从图上可以看出，与点 X9 相关联的 Voronoi 区域由连接到 X9 的 Delaunay 边的垂直平分线确定。此外，Voronoi 边的顶点位于 Delaunay 三角形的外接圆心处。通过绘出三角形 $\{X9, X4, X8\}$ 的外接圆心，可以说明这些联系。

为求出此三角形的索引，请查询三角剖分。此三角形包含位置 $(-1, 0)$ 。

```
tidx = pointLocation(dt,-1,0);
```

现在，求出此三角形的外接圆心，并用绿色绘图。

```
cc = circumcenter(dt,tidx);
hold on
plot(cc(1),cc(2),'*g');
hold off
```



Delaunay 三角剖分和 Voronoi 图互为几何对偶。从 Delaunay 三角剖分可以计算 Voronoi 图，反之亦然。

观察与凸包上的点相关联的 Voronoi 区域是否是无界的（例如与 X13 相关联的 Voronoi 区域）。此区域中的边“终止”于无穷远处。平分 Delaunay 边 (X13, X12) 和 (X13, X14) 的 Voronoi 边延伸至无穷远处。虽然 Voronoi 图提供了点集中每个点附近空间的最近邻点分解，但是不直接支持最近邻点查询。然而，用于计算 Voronoi 图的几何结构也用于执行最近邻点搜索。

计算 Voronoi 图

本示例显示如何计算二维和三维 Voronoi 图。

MATLAB 提供用于绘制二维 Voronoi 图以及计算 N 维 Voronoi 图的拓扑的函数。事实上，由于所需内存的指数级增长，对高于六维的大中型数据集，Voronoi 计算是不实际的。

voronoi 绘图函数为二维空间的点集绘制 Voronoi 图。在 MATLAB 中，有两种方式计算点集的 Voronoi 图：

- 使用 MATLAB 函数 **voronoin**
- 使用 **delaunayTriangulation** 方法 **voronoiDiagram**。

voronoin 函数支持为 N 维空间 ($N \geq 2$) 中的离散点计算 Voronoi 拓扑。**voronoiDiagram** 方法支持为二维或三维离散点计算 Voronoi 拓扑。

建议用 **voronoiDiagram** 方法计算二维或三维拓扑，因为此方法更稳定，对大型数据集表现更佳。此方法支持递增插值、移除点和补充查询，例如最近邻点搜索。

voronoin 函数和 **voronoiDiagram** 方法使用矩阵格式表示 Voronoi 图的拓扑。有关此数据结构的详细信息，请参阅“三角剖分矩阵格式”（第 7-3 页）。

给定点集 **X**，按如下方式获取 Voronoi 图的拓扑：

- 使用 **voronoin** 函数

```
[V,R] = voronoin(X)
```

- 使用 **voronoiDiagram** 方法

```
dt = delaunayTriangulation(X);
```

```
[V,R] = voronoiDiagram(dt)
```

V 是表示 Voronoi 顶点（这些顶点是 Voronoi 边的端点）坐标的矩阵。按照惯例，**V** 中的第一个顶点是无限顶点。**R** 是向量元胞数组长度 **size(X,1)**，表示与每个点相关联的 Voronoi 区域。因此，与点 **X(i)** 相关联的 Voronoi 区域是 **R{i}**。

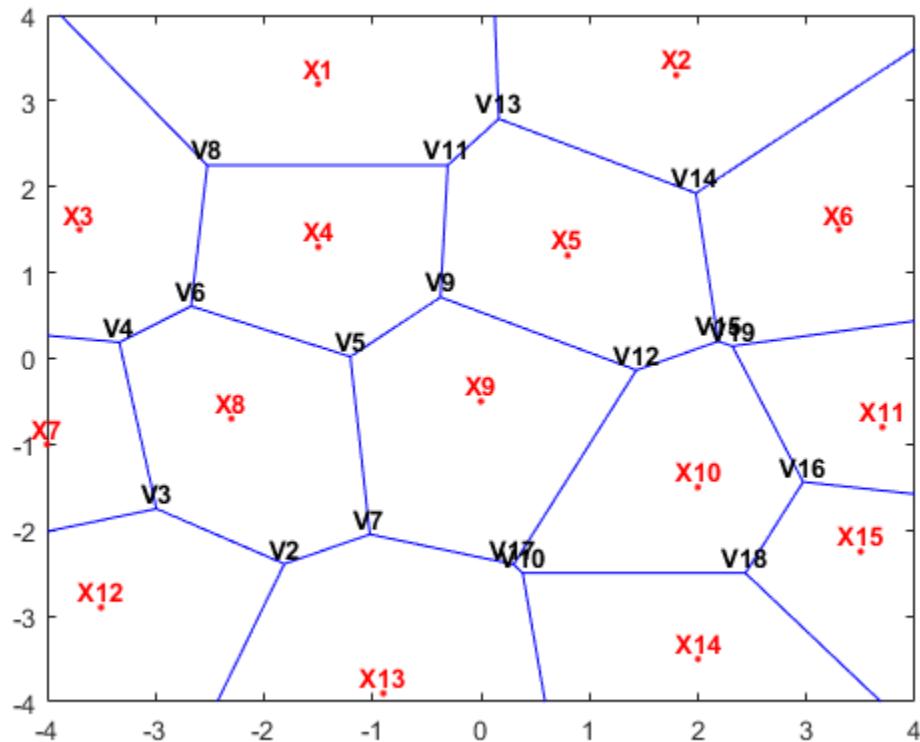
定义点集并绘制其 Voronoi 图

```
X = [-1.5 3.2; 1.8 3.3; -3.7 1.5; -1.5 1.3; 0.8 1.2; ...
      3.3 1.5; -4.0 -1.0; -2.3 -0.7; 0 -0.5; 2.0 -1.5; ...
      3.7 -0.8; -3.5 -2.9; -0.9 -3.9; 2.0 -3.5; 3.5 -2.25];
[VX,VY] = voronoi(X(:,1),X(:,2));
h = plot(VX,VY,'b',X(:,1),X(:,2),'r');
xlim([-4,4])
ylim([-4,4])

% Assign labels to the points X.
nump = size(X,1);
plabels = arrayfun(@(n) {sprintf('X%d', n)}, (1:nump)');
hold on
Hpl = text(X(:,1), X(:,2)+0.2, plabels, 'color', 'r', ...
            'FontWeight', 'bold', 'HorizontalAlignment',...
            'center', 'BackgroundColor', 'none');

% Compute the Voronoi diagram.
dt = delaunayTriangulation(X);
[V,R] = voronoiDiagram(dt);

% Assign labels to the Voronoi vertices V.
% By convention the first vertex is at infinity.
numv = size(V,1);
vlabels = arrayfun(@(n) {sprintf('V%d', n)}, (2:numv)');
hold on
Hpl = text(V(2:end,1), V(2:end,2)+.2, vlabels, ...
            'FontWeight', 'bold', 'HorizontalAlignment',...
            'center', 'BackgroundColor', 'none');
hold off
```



R{9} 给出了与点位 X9 相关联的 Voronoi 顶点的索引。

R{9}

ans = 1×5

5 7 17 12 9

Voronoi 顶点的索引是基于 V 数组的索引。

类似地, **R{4}** 给出了与点位 X4 相关联的 Voronoi 顶点的索引。

R{4}

ans = 1×5

5 9 11 8 6

在三维空间中, Voronoi 区域为凸多面体, 创建 Voronoi 图的语法是相似的。但 Voronoi 图的几何形状更加复杂。以下示例描述了三维 Voronoi 图的创建过程和单个区域的绘制过程。

在三维空间创建一个包含 25 个点的样本, 并计算此点集的 Voronoi 图的拓扑。

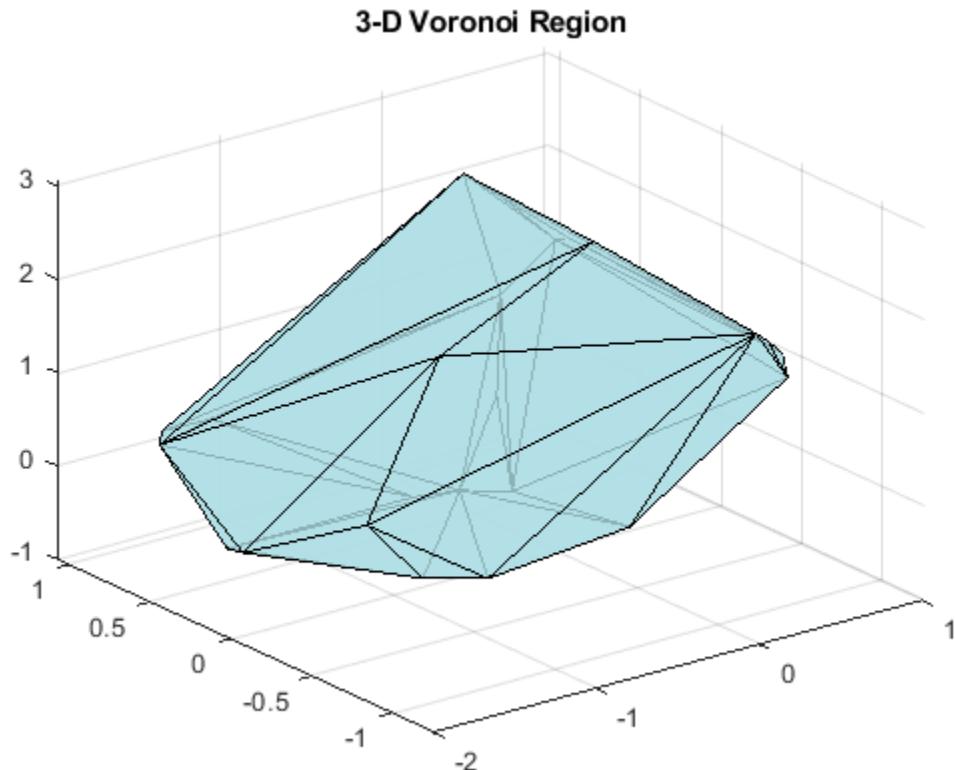
```
X = -3 + 6.*gallery('uniformdata',[25 3],0);
dt = delaunayTriangulation(X);
```

计算 Voronoi 图的拓扑。

```
[V,R] = voronoiDiagram(dt);
```

求距离原点最近的点，并绘制与此点相关联的 Voronoi 区域。

```
tid = nearestNeighbor(dt,0,0,0);
XR10 = V(R{tid},:);
K = convhull(XR10);
defaultFaceColor = [0.6875 0.8750 0.8984];
trisurf(K, XR10(:,1) ,XR10(:,2) ,XR10(:,3) , ...
    'FaceColor', defaultFaceColor, 'FaceAlpha',0.8)
title('3-D Voronoi Region')
```



另请参阅

详细信息

- “空间搜索”（第 7-51 页）

区域边界的类型

本节内容

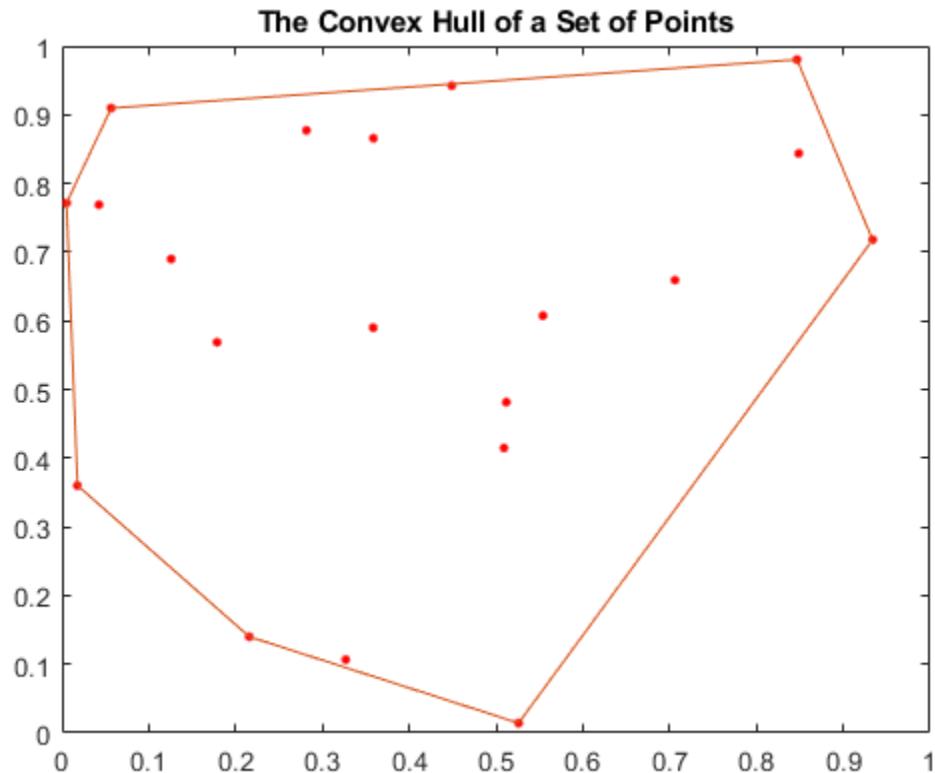
“凸包与非凸多边形”（第 7-64 页）

“阿尔法形状”（第 7-66 页）

凸包与非凸多边形

N 维空间中点集的凸包是封闭该点集中所有点的最小凸形区域。如果将二维点集想象为钉在木板上的钉子，则通过拉一根橡皮筋并围绕所有钉子将会构成该点集的凸包。

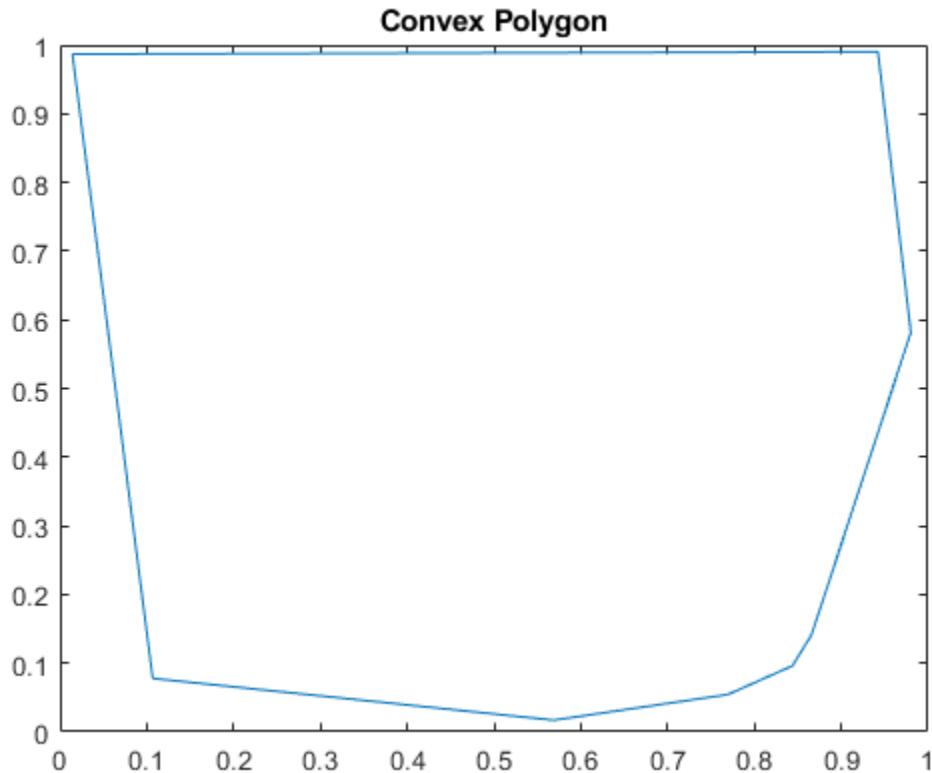
```
x = gallery('uniformdata',20,1,20);
y = gallery('uniformdata',20,1,30);
plot(x,y,'r!','MarkerSize',10)
hold on
k = convhull(x,y);
plot(x(k),y(k))
title('The Convex Hull of a Set of Points')
hold off
```



凸多边形是不含凹顶点的多边形，例如：

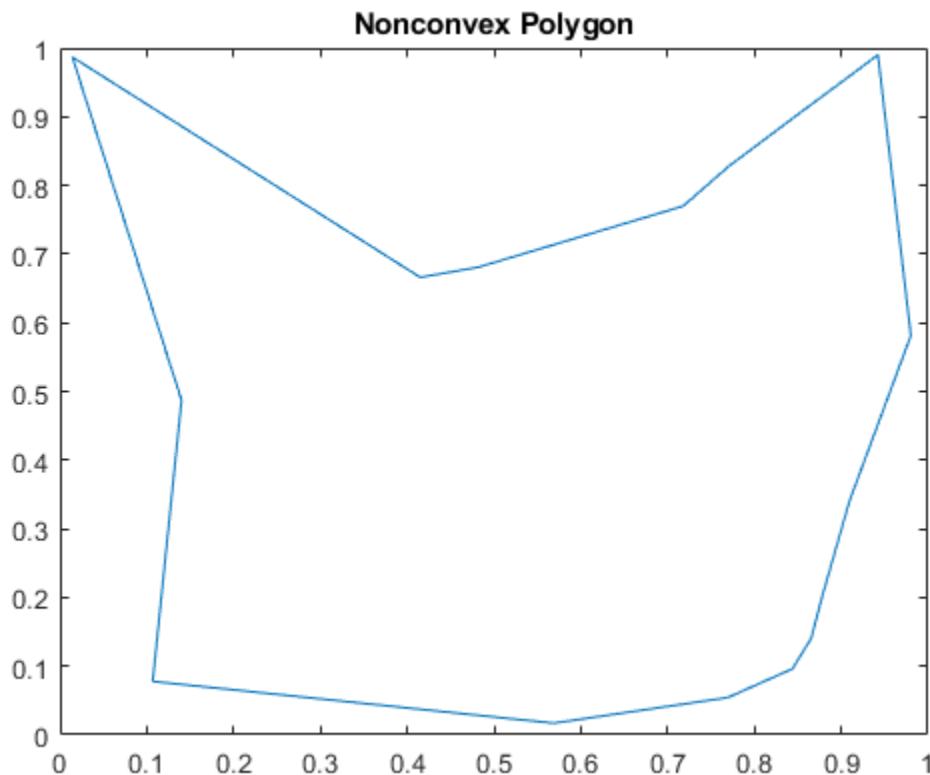
```
x = gallery('uniformdata',20,1,30);
y = gallery('uniformdata',20,1,40);
k = convhull(x,y);
```

```
plot(x(k),y(k))
title('Convex Polygon')
```



您也可以创建非凸点集的边界。如果从上面对凸包进行“真空包装”，您可以使用凹顶点将所有点封闭在非凸多边形中：

```
k = boundary(x,y,0.9);
plot(x(k),y(k))
title('Nonconvex Polygon')
```



凸包具有很多应用。从平面中离散点集的凸包可以计算由该点集确定边界的区域的上界。凸包简化了更复杂的多边形或多面体的表现形式。例如，要确定两个非凸几何体是否相交，可以应用一系列快速否定步骤避免进行完全相交分析造成的损失：

- 检查围绕每个几何体的轴对齐边界框是否相交。
- 如果边界框相交，则可以计算每个几何体的凸包，并检查凸包的交集。

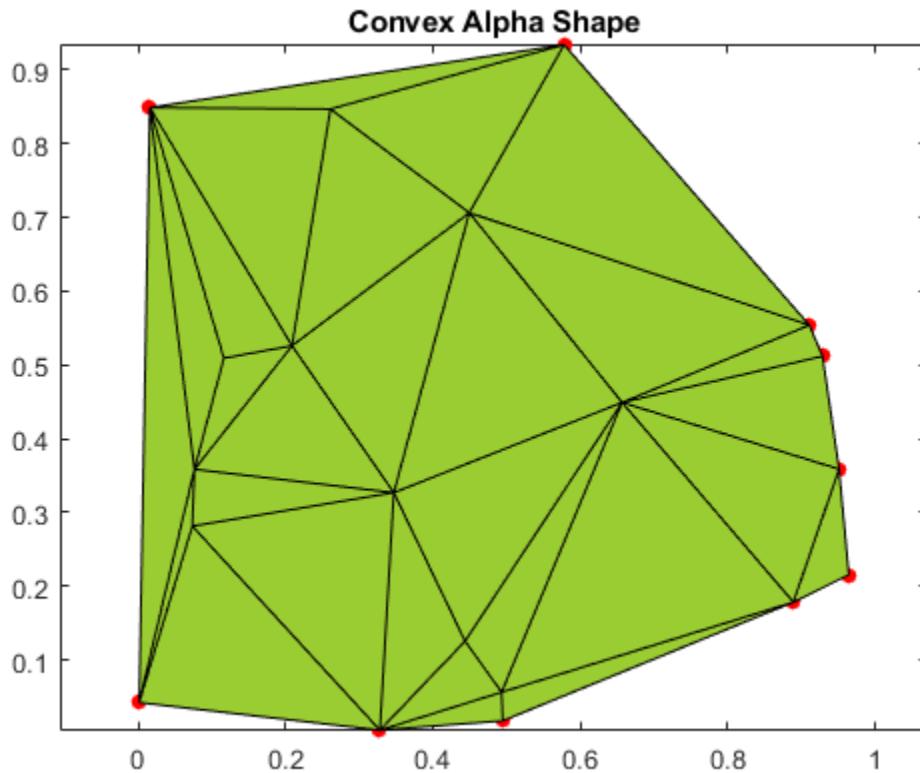
如果凸包不相交，则会避免进行更广泛的相交检验所需要的代价。

尽管凸包和非凸多边形是表示相对简单边界的方便方法，但它们实际是一种称为阿尔法形状的更常见几何结构的特定实例。

阿尔法形状

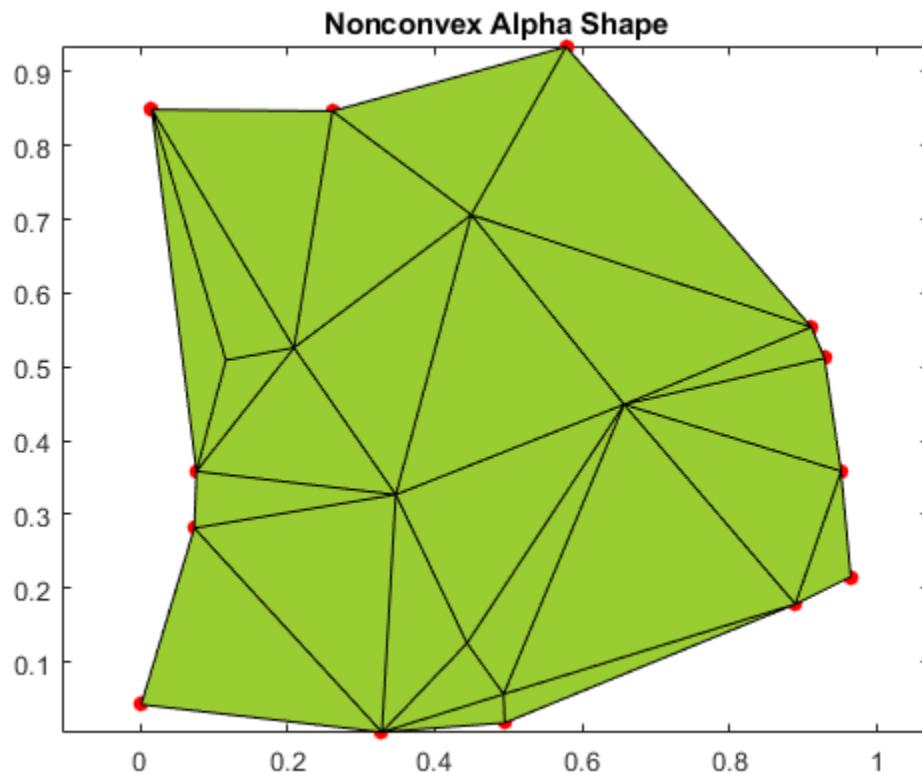
点集的阿尔法形状是凸包和 Delaunay 三角剖分子图的推广。换言之，凸包只是其中一种阿尔法形状类型，完整系列的阿尔法形状可以从给定点集的 Delaunay 三角剖分进行派生。

```
x = gallery('uniformdata',20,1,10);
y = gallery('uniformdata',20,1,20);
plot(x,y,'r!','MarkerSize',20)
hold on
shp = alphaShape(x,y,100);
plot(shp)
title('Convex Alpha Shape')
hold off
```



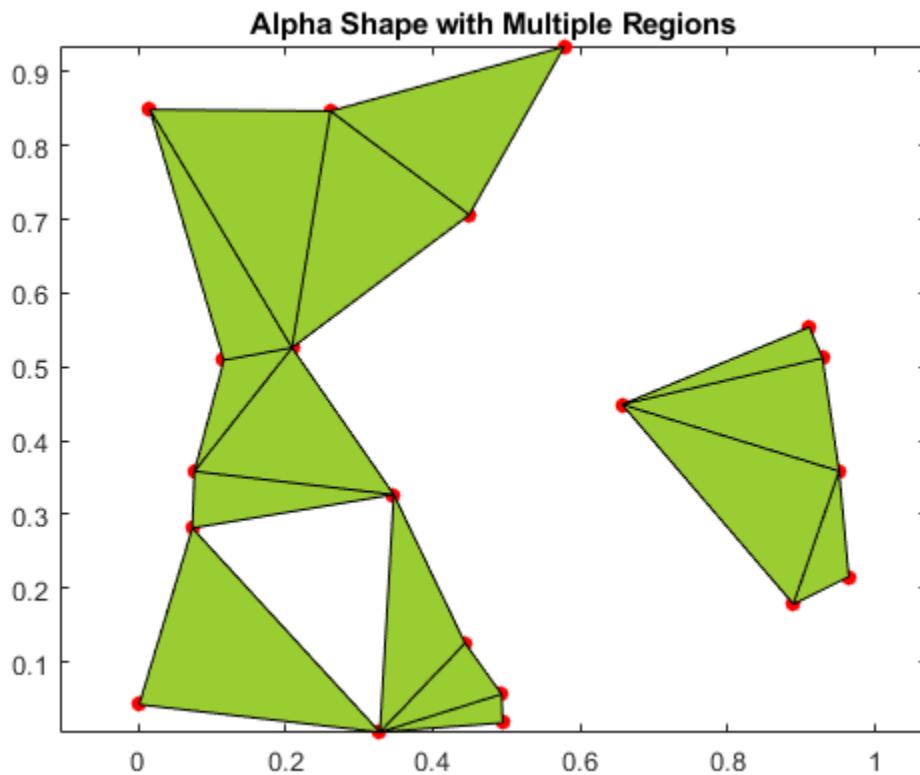
与凸包不同，阿尔法形状具有用于控制细节程度或者边界在点集周围的拟合紧密度的参数。该参数称为阿尔法或阿尔法半径。在 0 到 Inf 范围内更改 alpha 半径可生成一组该点集特有的不同 alpha 形状。

```
plot(x,y,'r!','MarkerSize',20)
hold on
shp = alphaShape(x,y,.5);
plot(shp)
title('Nonconvex Alpha Shape')
hold off
```



更改阿尔法半径有时会产生具有多个区域的阿尔法形状，其中可能包含也可能不包含孔洞。不过，MATLAB® 中的 `alphaShape` 函数始终返回正则化的 alpha 形状，以防出现孤立或空悬的点、边或面。

```
plot(x,y,'r','MarkerSize',20)
hold on
shp = alphaShape(x,y);
plot(shp)
title('Alpha Shape with Multiple Regions')
hold off
```



另请参阅

[alphaShape](#) | [convhull](#)

详细信息

- “使用 delaunayTriangulation 类” (第 7-19 页)
- “三角剖分矩阵格式” (第 7-3 页)

计算凸包

本节内容

- “使用 `convhull` 和 `convhulln` 计算凸包” (第 7-70 页)
- “使用 `delaunayTriangulation` 类计算凸包” (第 7-73 页)
- “使用 `alphaShape` 的凸包计算” (第 7-74 页)

MATLAB 提供多种计算凸包的方式：

- 使用 MATLAB 函数 `convhull` 和 `convhulln`
- 使用 `delaunayTriangulation` 类提供的 `convexHull` 方法
- 使用 `alphaShape` 函数以及 `alpha` 半径 `Inf`。

`convhull` 函数支持在二维和三维空间中计算凸包。`convhulln` 函数支持在 N 维空间 ($N \geq 2$) 中计算凸包。对于二维或三维计算，建议使用 `convhull` 函数，因为其稳定性和性能更好。

`delaunayTriangulation` 类支持从 Delaunay 三角剖分进行凸包的二维或三维计算。这种计算方式的效率不如专用的 `convhull` 和 `convhulln` 函数。但是，如果有一个点集的 `delaunayTriangulation`，并且需要凸包，则 `convexHull` 方法可从现有的三角剖分更高效地计算凸包。

`alphaShape` 函数还通过将 `alpha` 半径输入参数设置为 `Inf`，来支持凸包的二维或三维计算。但是，与 `delaunayTriangulation` 相似，使用 `alphaShape` 计算凸包不如直接使用 `convhull` 或 `convhulln` 高效。在处理以前创建的阿尔法形状对象时例外。

使用 `convhull` 和 `convhulln` 计算凸包

`convhull` 和 `convhulln` 函数取一个点集，输出位于凸包边界上的点的索引。凸包基于点索引的表示法支持绘图，且便于数据访问。下面这些示例说明凸包的计算和表示方式。

第一个示例使用 `seamount` 数据集的一个二维点集作为 `convhull` 函数的输入。

加载数据。

```
load seamount
```

计算点集的凸包。

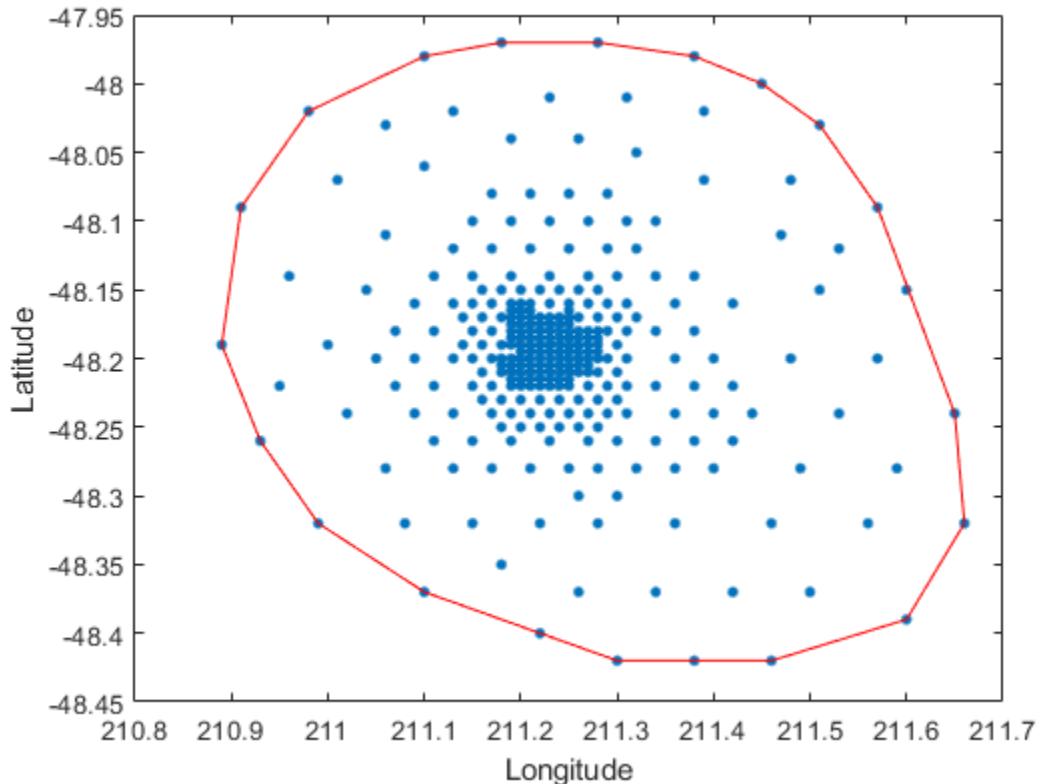
```
K = convhull(x,y);
```

`K` 表示绕凸包沿逆时针方向排列的点的索引。

绘制数据及其凸包。

```
plot(x,y,'','markersize',12)
xlabel('Longitude')
ylabel('Latitude')
hold on
```

```
plot(x(K),y(K),'r')
```



给凸包上的点添加点标签，以观察 K 的结构。

```
[K,A] = convhull(x,y);
```

convhull 可以计算二维和三维点集的凸包。可以重复使用海底山数据集说明三维凸包的计算。

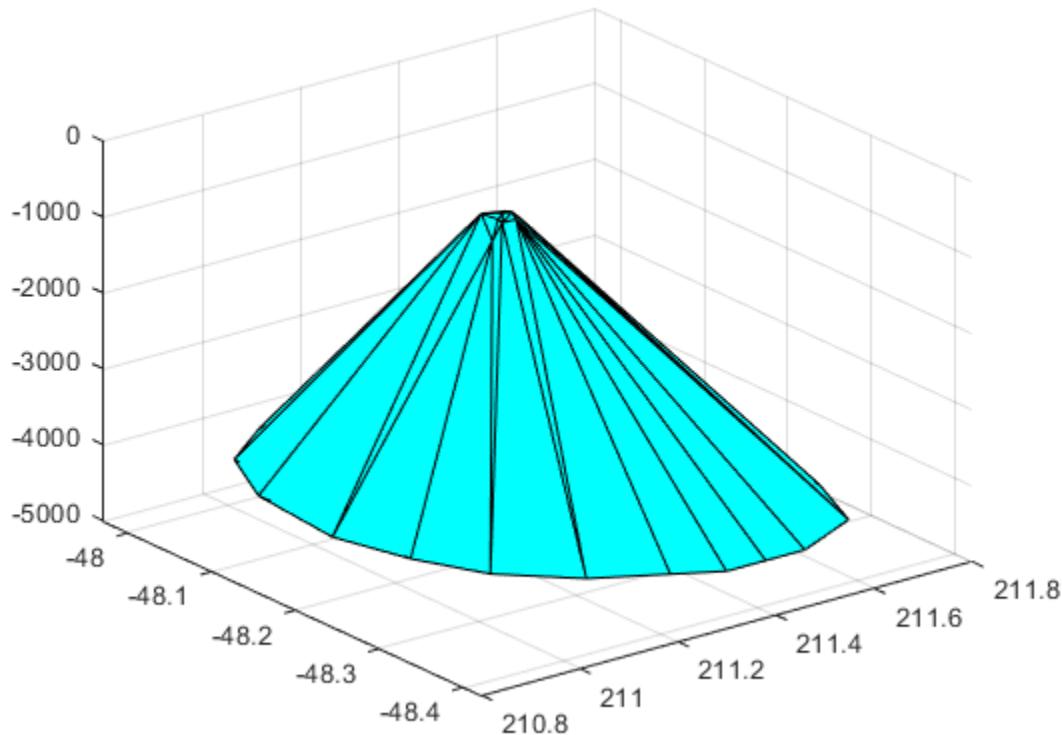
包含海底山 z 坐标数据仰角。

```
close(gcf)
K = convhull(x,y,z);
```

在三维空间中，凸包的边界 K 由三角剖分表示。这是以矩阵格式表示的一组三角面，并有关于点数组的索引。矩阵 K 的每一行表示一个三角形。

由于凸包的边界表示为三角剖分，因此可以使用三角剖分绘图函数 **trisurf**。

```
trisurf(K,x,y,z,'Facecolor','cyan')
```



三维凸包约束的体积可由 **convhull** 选择性返回，语法如下。

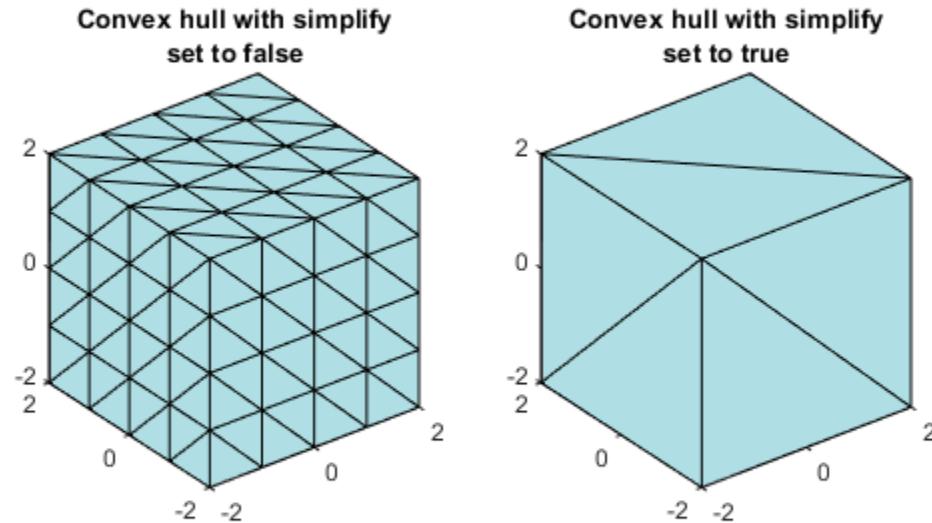
```
[K,V] = convhull(x,y,z);
```

convhull 函数还提供通过移除对面积或体积无贡献的顶点来简化凸包表示的选项。例如，如果凸包的边界面共线或共面，则可以合并这些面，以给出更精确的表示。下面的示例说明此选项的用法。

```
[x,y,z] = meshgrid(-2:1:2,-2:1:2,-2:1:2);
x = x(:);
y = y(:);
z = z(:);

K1 = convhull(x,y,z);
subplot(1,2,1)
defaultFaceColor = [0.6875 0.8750 0.8984];
trisurf(K1,x,y,z,'Facecolor',defaultFaceColor)
axis equal
title(sprintf('Convex hull with simplify\nset to false'))

K2 = convhull(x,y,z,'simplify',true);
subplot(1,2,2)
trisurf(K2,x,y,z,'Facecolor',defaultFaceColor)
axis equal
title(sprintf('Convex hull with simplify\nset to true'))
```



MATLAB 提供 `convhulln` 函数来支持更高维凸包和超体积的计算。虽然 `convhulln` 支持 N 维，但是由于内存需求迅猛增长，对高于十维情况下出现的问题难以应对。

`convhull` 函数在二维和三维情况下优于 `convhulln`，因为它更稳定，表现出更佳的性能。

使用 `delaunayTriangulation` 类计算凸包

本示例显示在二维空间中点集的 Delaunay 三角剖分与该点集的凸包之间的关系。

`delaunayTriangulation` 类支持二维和三维空间中 Delaunay 三角剖分的计算。这个类还提供 `convexHull` 方法从三角剖分获得凸包。

在二维空间内创建一组点的 Delaunay 三角剖分。

```
X = [-1.5 3.2; 1.8 3.3; -3.7 1.5; -1.5 1.3; 0.8 1.2; ...
    3.3 1.5; -4.0 -1.0; -2.3 -0.7; 0 -0.5; 2.0 -1.5; ...
    3.7 -0.8; -3.5 -2.9; -0.9 -3.9; 2.0 -3.5; 3.5 -2.25];
```

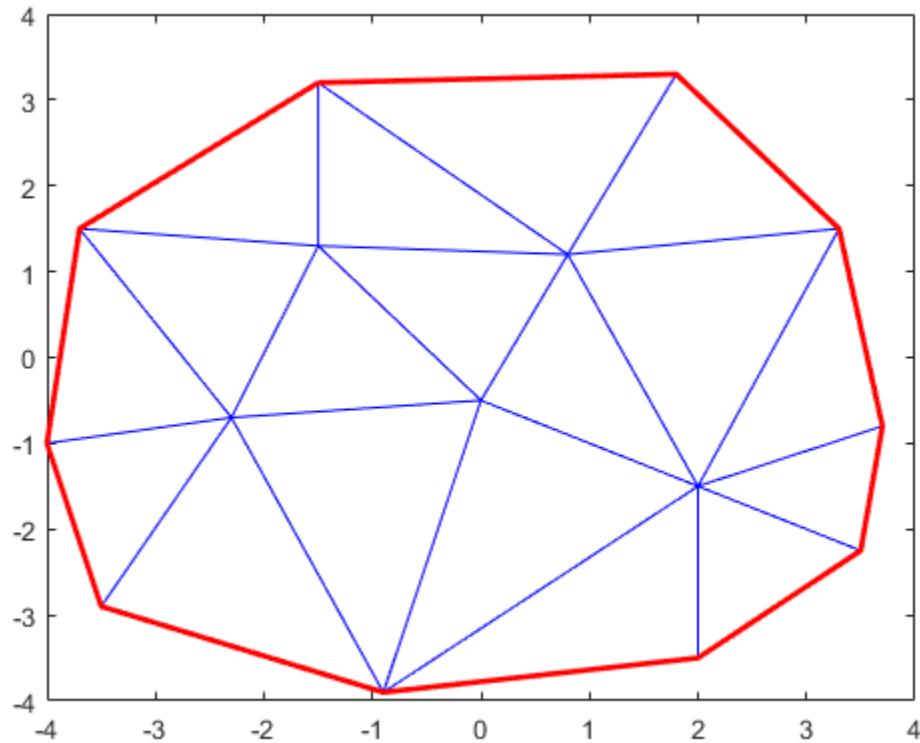
```
dt = delaunayTriangulation(X);
```

绘制三角剖分并高亮显示仅由展现该凸包的单一三角形共有的边。

```
triplot(dt)
```

```
fe = freeBoundary(dt)';
hold on
```

```
plot(X(fe,1), X(fe,2), '-r', 'LineWidth',2)
hold off
```



在三维空间中，仅由一个四面体共有的三角剖分的面表示凸包的边界。

专用 `convhull` 函数通常比基于 `convexHull` 方法的计算更高效。但是，在以下情况下适合基于三角剖分的途径：

- 已有点集的一个 `delaunayTriangulation`，并且还需要凸包。
- 需要逐渐从点集中添加或移除一些点，并且需要在编辑这些点后频繁地重新计算凸包。

使用 `alphaShape` 的凸包计算

此示例说明了如何使用 `alphaShape` 函数计算二维点集的凸包。

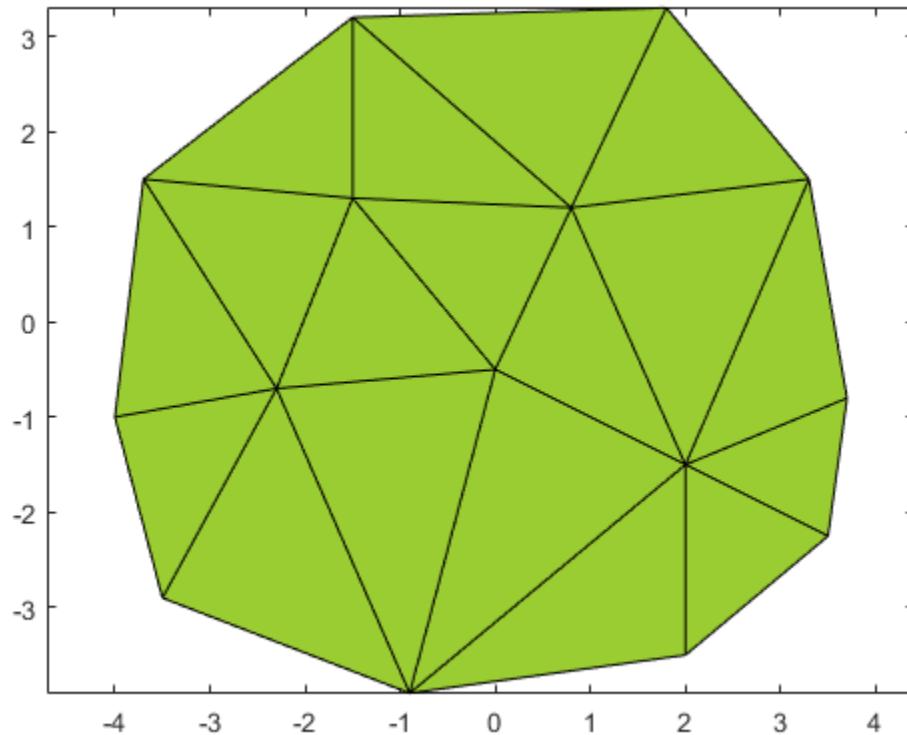
`alphaShape` 通过二维或三维点集计算正则化的 alpha 形状。您可以指定阿尔法半径，它确定阿尔法形状包围点集的松紧程度。在 alpha 半径设为 Inf 时，生成的 alpha 形状是点集的凸包。

创建一个二维点集。

```
X = [-1.5 3.2; 1.8 3.3; -3.7 1.5; -1.5 1.3; 0.8 1.2; ...
      3.3 1.5; -4.0 -1.0; -2.3 -0.7; 0 -0.5; 2.0 -1.5; ...
      3.7 -0.8; -3.5 -2.9; -0.9 -3.9; 2.0 -3.5; 3.5 -2.25];
```

使用 alpha 半径等于 Inf 的 alpha 形状计算并绘制该点集的凸包。

```
shp = alphaShape(X,Inf);
plot(shp)
```



另请参阅

[alphaShape](#) | [convexHull](#) | [convhull](#) | [convhulln](#) | [delaunayTriangulation](#)

相关示例

- “使用 Delaunay 三角剖分” (第 7-13 页)

插值

- “网格和散点样本数据” (第 8-2 页)
- “插入网格数据” (第 8-3 页)
- “多个一维值集的插值” (第 8-33 页)
- “将二维选择项插入三维网格中” (第 8-35 页)
- “内插散点数据” (第 8-37 页)
- “使用特定 Delaunay 三角剖分的插值” (第 8-59 页)
- “外插散点数据” (第 8-62 页)
- “对不同量级的数据进行归一化” (第 8-68 页)
- “使用网格插值对图像重采样” (第 8-71 页)

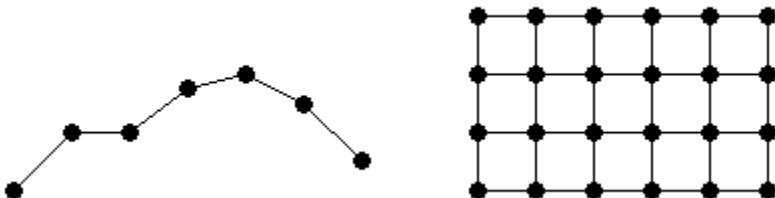
网格和散点样本数据

插值是在位于一组样本数据点域中的查询位置进行数值估算的方法。可插入一个由位置 X 和对应值 V 定义的样本数据集，以产生一个形式为 $V = F(X)$ 的函数。然后可以使用此函数为查询点 X_q 求值，给出 $V_q = F(X_q)$ 。这是一个单值函数；对于 X 的定义域中的任何查询 X_q ，它将产生唯一的值 V_q 。为了产生满意的插值，假定样本数据适用此属性。另一个有趣的特性是插值函数穿过数据点。这是插值与曲线/曲面拟合的一个重要区别。在拟合中，函数不必穿过样本数据点。

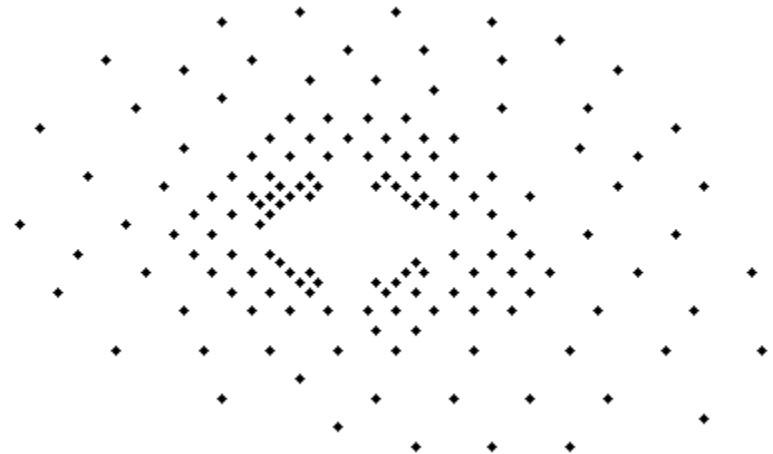
值 V_q 的计算通常基于查询点 X_q 的邻点中的数据点。有许多执行插值的方式。在 MATLAB 中，根据样本数据的结构，插值分为两类。在对齐轴的网格中，可对样本数据排序，否则它们可能会分散。在样本点由网格分布的情况下，可以利用数据的组织结构有效求出查询邻点中的样本点。另一方面散点数据的插值需要数据点的三角剖分，这就产生了附加计算标准。

以下部分讨论了两种插值方式：

- “插入网格数据”（第 8-3 页）部分讨论轴对齐网格格式的样本数据的一维插值和 N 维 ($N \geq 2$) 插值：



- “内插散点数据”（第 8-37 页）部分讨论散点数据的 N 维 ($N \geq 2$) 插值。



插入网格数据

本节内容

- “网格数据表示” (第 8-3 页)
- “基于网格的插值” (第 8-10 页)
- “interp 系列函数的插值” (第 8-15 页)
- “用 griddedInterpolant 类插值” (第 8-24 页)

网格数据表示

- “网格表示” (第 8-3 页)
- “网格表示的类型” (第 8-8 页)
- “网格逼近技术” (第 8-9 页)

网格表示

此示例说明如何使用 `meshgrid` 和 `ndgrid` 创建二维网格。

在 MATLAB® 中，网格数据表示网格中的有序数据。要理解有序数据，您可以思考 MATLAB 在矩阵中存储数据的方式。

定义一些数据。

```
A = gallery('uniformdata',[3 5],0)
```

`A = 3×5`

```
0.9501 0.4860 0.4565 0.4447 0.9218
0.2311 0.8913 0.0185 0.6154 0.7382
0.6068 0.7621 0.8214 0.7919 0.1763
```

MATLAB 在矩阵中存储数据。可以将 `A` 视为一组按矩阵索引排序的元素位置。`A` 的线性索引为：

$$\begin{bmatrix} 1 & 4 & 7 & 10 & 13 \\ 2 & 5 & 8 & 11 & 14 \\ 3 & 6 & 9 & 12 & 15 \end{bmatrix}$$

可通过索引来检索矩阵中的任何元素，即请求矩阵中该位置上的元素。通过 `A(i)` 可检索 `A` 中的第 `i` 个元素。

检索 `A` 中的第 7 个元素。

`A(7)`

```
ans = 0.4565
```

对于 $m \times n$ 矩阵，可通过将 `i` 偏移 1 位来求与第 `i` 个元素相邻的列元素。若要求与第 `i` 个元素相邻的行元素，需将 `i` 偏移 `m` 位。

$$\begin{array}{ccccc} & & i - 1 & & \\ i - m & i & i & i + m & \\ & & i + 1 & & \end{array}$$

检索与 A(7) 相邻的列元素。

A(6),A(8)

```
ans = 0.7621
```

```
ans = 0.0185
```

MATLAB 使用类似的思路创建数据网格。网格不只是一个符合特定几何属性的点集，更是一个依赖于网格中的点之间的有序关系的网格数据集。网格结构体中随时可用的相邻信息对于许多应用（尤其是基于网格的插值）非常有用。

MATLAB 提供了两个用于创建网格的函数：

- **meshgrid** 创建与笛卡尔轴对齐的二维和三维网格。创建二维网格的语法为 $[X,Y] = \text{meshgrid}(xgv, ygv)$ ，其中 xgv 是长度为 m 的向量， ygv 是长度为 n 的向量。**meshgrid** 通过复制 xgv 构成 $n \times m$ 矩阵 X ，并通过复制 ygv 构成另一个 $n \times m$ 矩阵 Y 。 X 和 Y 表示网格点的坐标。 X 的行与水平 X 轴对齐， Y 的列与负 Y 轴对齐。
- **ndgrid** 创建与数组空间对齐的 N 维网格。在数组空间中，坐标区为行、列、页面等。调用语法为 $[X1, X2, X3, \dots, Xn] = \text{ndgrid}(x1gv, x2gv, x3gv, \dots, xngv)$ ，其中 $x1gv, x2gv, x3gv, \dots, xngv$ 是在各个维度涵盖网格的向量， $X1, X2, X3, \dots, Xn$ 是可用于对多变量函数求值和用于多维插值的输出数组。

使用 **meshgrid** 可从两个向量 xgv 和 ygv 创建与二维轴对齐的网格。

```
xgv = [1 2 3];
ygv = [1 2 3 4 5];
[X,Y] = meshgrid(xgv, ygv)
```

$X = 5 \times 3$

```
1   2   3
1   2   3
1   2   3
1   2   3
1   2   3
```

$Y = 5 \times 3$

```
1   1   1
2   2   2
3   3   3
4   4   4
5   5   5
```

现在使用 **ndgrid** 从相同的两个向量 xgv 和 ygv 创建与二维空间对齐的网格。

```
[X1,X2] = ndgrid(xgv,ygv)
```

$X1 = 3 \times 5$

```
1   1   1   1   1
2   2   2   2   2
3   3   3   3   3
```

$X2 = 3 \times 5$

```

1   2   3   4   5
1   2   3   4   5
1   2   3   4   5

```

请注意，`ndgrid` 的 `X1` 是 `meshgrid` 的 `X` 的转置。`X2` 和 `Y` 同样如此。

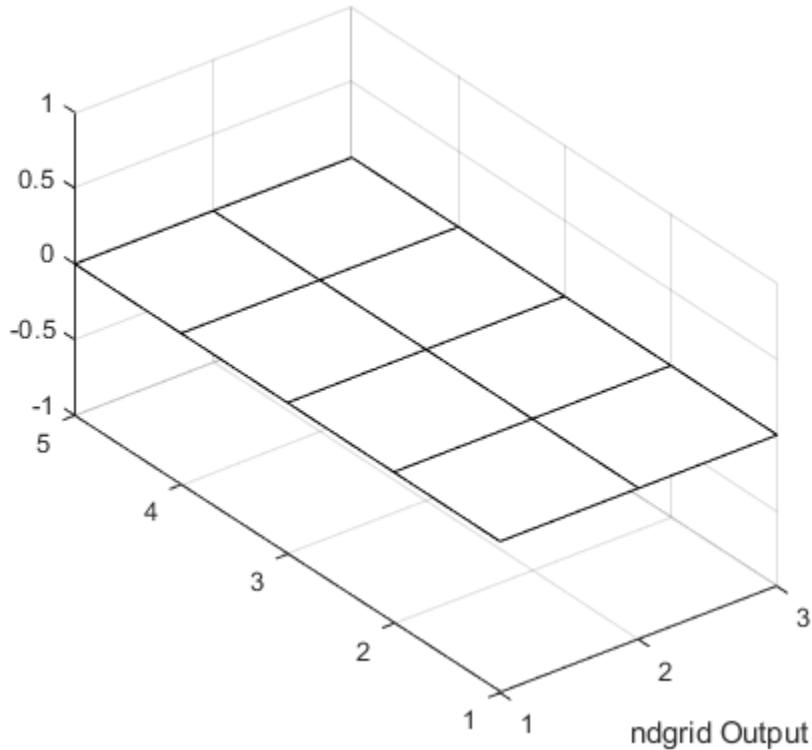
对于给定的输入集，`meshgrid` 和 `ndgrid` 函数将生成具有完全相同的坐标的网格。它们的输出之间的唯一差别是坐标数组的格式。绘制两个输出，可以看到它们是相同的。

```

figure()
[X1_ndgrid,X2_ndgrid] = ndgrid(1:3,1:5);
Z = zeros(3,5);
mesh(X1_ndgrid,X2_ndgrid,Z,'EdgeColor','black')
axis equal;

% Set the axis labeling and title
h1 = gca;
h1.XTick = [1 2 3];
h1.YTick = [1 2 3 4 5];
xlabel('ndgrid Output')

```



```

figure()
[X_meshgrid,Y_meshgrid] = meshgrid(1:3, 1:5);
mesh(X_meshgrid,Y_meshgrid,Z,'EdgeColor','black')
axis equal;

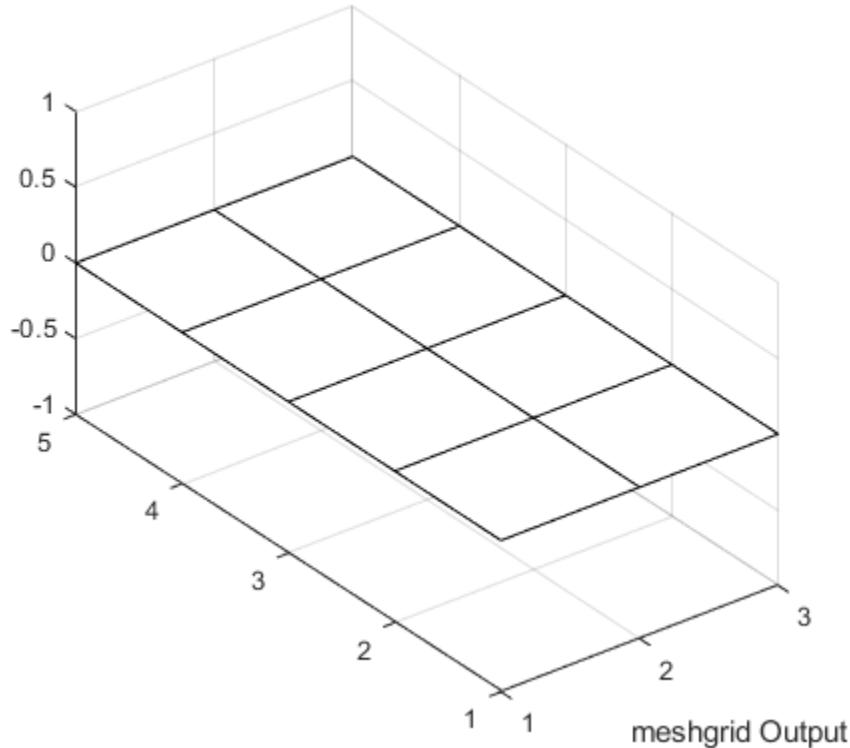
% Set the axis labeling and title

```

```

h2 = gca;
h2.XTick = [1 2 3];
h2.YTick = [1 2 3 4 5];
xlabel('meshgrid Output')

```



根据网格的用途，可以选择其中的一种格式。MATLAB 中的一些函数可能要求您的数据采用 **meshgrid** 格式，而另一些函数可能要求 **ndgrid** 格式。

网格格式间的转换

要将一个二维网格输出从 **meshgrid** 转换为 **ndgrid** 格式，请转置坐标矩阵：

```

[X_meshgrid,Y_meshgrid] = meshgrid(1:3, 1:5);
[X1_ndgrid,X2_ndgrid] = ndgrid(1:3,1:5);

isequal(X_meshgrid',X1_ndgrid)
ans =
    1
isequal(Y_meshgrid',X2_ndgrid)
ans =
    1

```

也可以使用 **permute** 函数。

```

isequal(permute(X_meshgrid,[2 1]),X1_ndgrid)
ans =
    1

```

要将一个三维 `meshgrid` 转换为 `ndgrid`, 请转置坐标数组的每一页。对于给定的数组 `my_array`, `permute(my_array, [2 1 3])` 进行行列互换, 实际效果是对每一页进行转置:

```
[X_meshgrid,Y_meshgrid,Z_meshgrid] = meshgrid(1:3, 1:5, [1 2]);
[X1_ndgrid,X2_ndgrid,X3_ndgrid] = ndgrid(1:3,1:5, [1 2]);
```

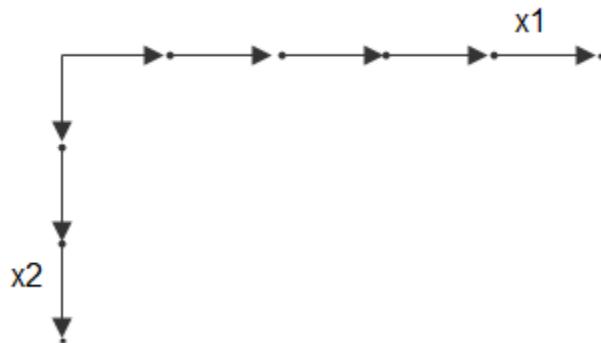
```
isequal(permute(X_meshgrid,[2 1 3]),X1_ndgrid)
ans =
1
```

```
isequal(permute(Y_meshgrid,[2 1 3]),X2_ndgrid)
ans =
1
```

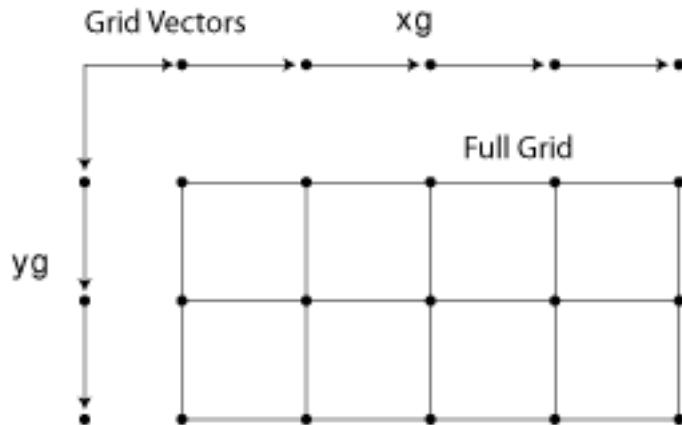
```
isequal(permute(Z_meshgrid,[2 1 3]),X3_ndgrid)
ans =
1
```

网格向量

传递给网格函数的输入称为网格向量。网格向量隐式定义了网格。以两个向量 `x1gv = (1:3)` 和 `x2gv = (1:5)` 为例。可以将这些向量视为沿 `x1` 方向的一组坐标和沿 `x2` 方向的一组坐标, 如下所示:



每个箭头指向一个位置。使用这两个向量可以定义一组网格点, 其中一组坐标由 `x1gv` 给定, 另一组坐标由 `x2gv` 给定。在复制网格向量时, 它们构成组成完整网格的两个坐标数组:



单调和非单调网格

输入的网格向量可能是单调或非单调的。单调向量所含的值在该维中递增，或在该维中递减。反之，非单调向量含有的值会上下波动。如果输入网格向量是非单调的，例如 [2 4 6 8 3 1]，则 `ndgrid` 输出以下结果：

```
[X1,X2] = ndgrid([2 4 6 8 3 1])
```

```
X1 =
 2 2 2 2 2
 4 4 4 4 4
 6 6 6 6 6
 3 3 3 3 3
 1 1 1 1 1
```

```
X2 =
 2 4 6 3 1
 2 4 6 3 1
 2 4 6 3 1
 2 4 6 3 1
 2 4 6 3 1
```

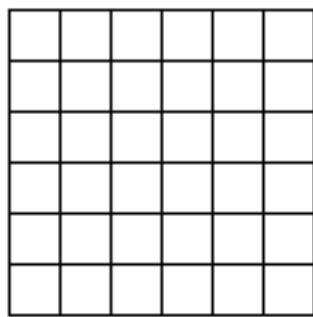
如果要将网格传递给其他 MATLAB 函数，网格向量应当是单调的。

均匀网格和非均匀网格

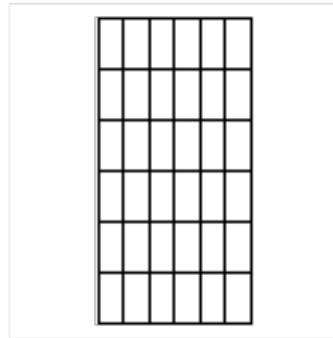
均匀网格是给定维度中所有邻点均有相等间距的网格。例如，`[X1, X2] = ndgrid([1 3 5 9],[11 13 15])` 是每一维度中间距均为两个单位的均匀网格。

没有必要让均匀网格中的间距在所有维度中均相等。例如，即使 `X1` 和 `X2` 中的间距不同，`[X1, X2] = ndgrid([1 2 3 4],[11 13 15])` 也被视为均匀网格。

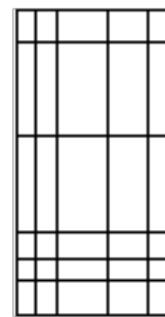
任一维度中存在间距不一致的网格即会称为非均匀网格。例如，`[X1, X2] = ndgrid([1 5 6 9],[11 13 15])` 便创建了一个非均匀网格，因为其间距沿第一维度变化。



均匀



均匀



非均匀

网格表示的类型

MATLAB 允许按以下三种方式之一表示网格：完整网格、简洁网格或默认网格。简洁网格和默认网格主要是为了方便使用和提高效率。

完整网格

完整网格是一种以显式方式定义点的网格。`ndgrid` 和 `meshgrid` 的输出定义一个完整网格。

简洁网格

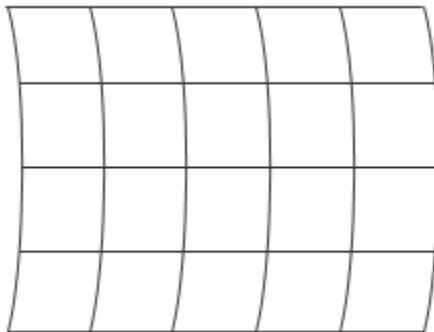
网格中每个点的显式定义与内存的消耗密切相关。简洁网格表示法无需完整网格的内存使用量。简洁网格表示法只存储网格向量，而不存储完整网格。（有关 `griddedInterpolant` 类如何使用简洁网格表示法的信息，请参阅“用 `griddedInterpolant` 类插值”（第 8-24 页）。）

默认网格

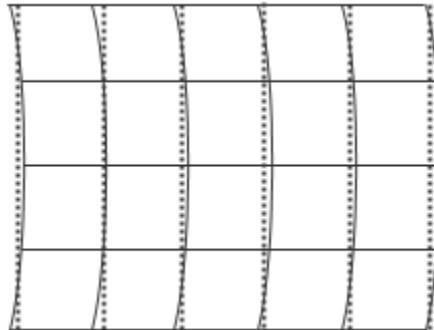
在许多情况下分析数据时，既关注网格中点之间的距离，也关注点的值。例如，一个数据集可能表示某一地理区域中特定点的降雨量。这种情况下，既关注每个网格点的值，也关注点与其相邻点之间的距离。其他情况下，可能只关注给定点的值，而不关注相对距离。例如，在处理来自核磁共振成像 (MRI) 扫描的输入数据时，其中的点之间的距离是完全均匀的。这种情况下，只会关注点的值，而确信完全均匀的网格。这种情况下，默认网格表示法可派上用场。默认网格表示法将显式存储网格点处的值，隐式创建网格点的坐标。

网格逼近技术

在某些情况下，可能需要数据的逼近网格。通过选择一组合适的网格向量，可由标准的 MATLAB 网格得到理想化的逼近网格。例如，网格具有的点可以位于曲线上。如果数据是基于经纬度的，则可能出现这样的数据集：



这种情况下，虽然输入数据无法直接网格化，但是可以在合适的区间逼近网格直线：



也可以使用默认网格（第 8-9 页）。

简并网格

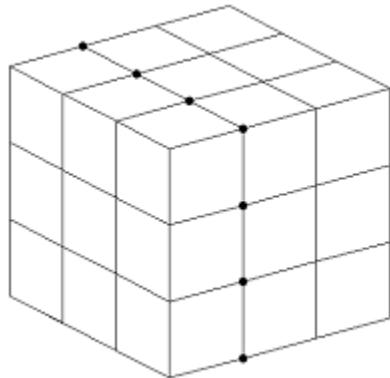
简并网格是网格的一维或多维是单一维度的网格特例。单一维度可以是内部的，如本示例中的 7:7:

```
[X1,X2,X3] = ndgrid(1:2:10,7:7,1:3:15);
```

单一维度也可以是后继维度：

```
[X1,X2,X3] = ndgrid(1:2:10,1:3:15,7:7);
```

如果要尝试取更大数据集的切片，可以创建一个简并网格。例如，您可能只想要分析三维 MRI 扫描的一个切片。这种情况下，需要来自多维网格的一个数据切片，例如以下图窗中的点式切片：



如果使用索引提取所需要的数据，则产生的网格是 X3 维中简并的网格：

```
[X1,X2,X3] = ndgrid(1:3);
```

```
X1_slice = X1(:,:,2)
X1_slice =
    1   1   1
    2   2   2
    3   3   3
```

```
X2_slice = X2(:,:,2)
X2_slice =
    1   2   3
    1   2   3
    1   2   3
```

```
X3_slice = X3(:,:,2)
X3_slice =
    2   2   2
    2   2   2
    2   2   2
```

数据网格化的概念对于理解 MATLAB 进行基于网格的插值的方式非常重要。

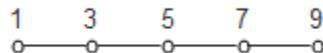
基于网格的插值

- “使用基于网格插值的好处”（第 8-11 页）
- “插值与拟合”（第 8-12 页）
- “插值方法”（第 8-12 页）

在基于网格的插值中，待插入的数据由有序网格表示。例如，准备在一个矩形平面曲面上以 1 厘米间隔垂直方向自顶向下、水平方向从左向右进行温度测量，视为二维网格化数据。基于网格的插值提供了获得网格点之间任意位置的温度的一种有效途径。

使用基于网格插值的好处

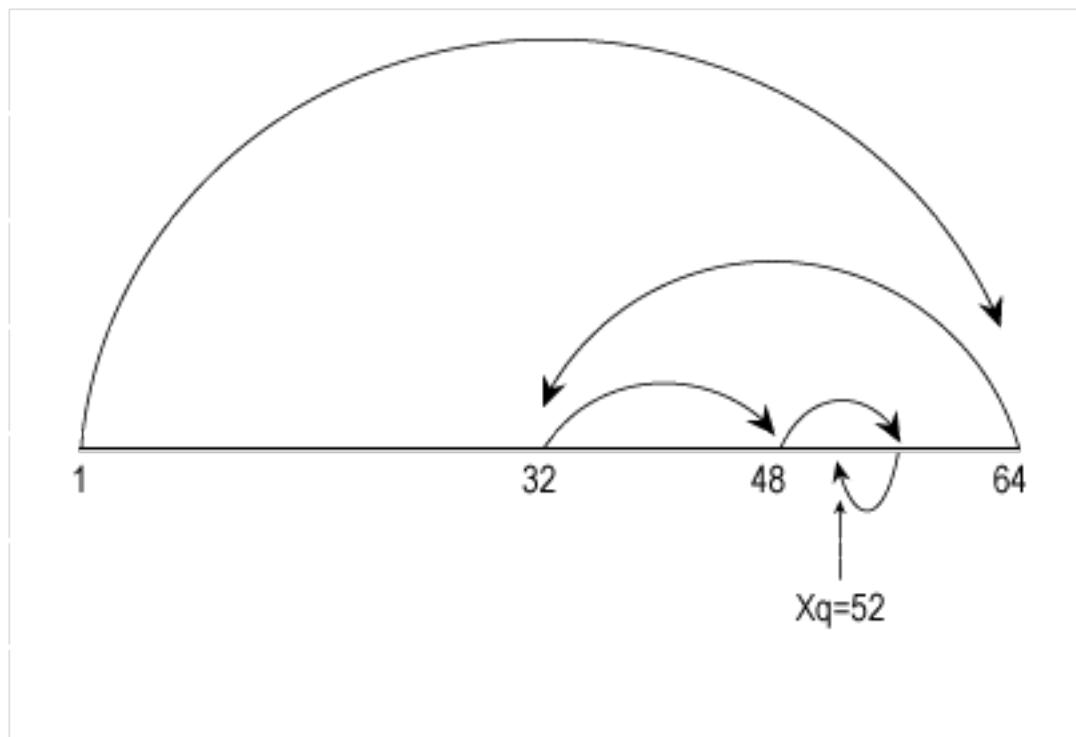
基于网格的插值大大节省了计算开销，因为网格化结构允许 MATLAB 非常迅速地定位查询点及其附近的相邻点。为了解其工作原理，以下面一维网格的点为例：



连接相邻点的直线表示网格的单元。第一个单元出现在 $x = 1$ 和 $x = 3$ 之间，第二个出现在 $x = 3$ 和 $x = 5$ 之间，依此类推。每个数表示网格中的一个坐标。如果要查询位于 $x = 6$ 处的网格，必须使用插值，因为网格中未显式定义 6。由于此网格具有均匀间距 2，因此可以用一次整除 ($6/2 = 3$) 缩小查询点的位置范围。这就说明该点位于网格的第三个单元中。定位二维网格中的单元涉及到在每一维中执行一次此操作。此操作称为快速查找，仅当数据分布在均匀网格中时，MATLAB 才使用这项技术。

这种快速查找高效定位含有查询点 X_q 的单元。二分查找按如下方式进行：

- 1 定位网格中心点。
- 2 将 X_q 与位于网格中心的点对比。
- 3 如果 X_q 小于在中心找到的点，则从搜索中排除所有大于中心点的网格点。同样，如果 X_q 大于在中心找到的点，则排除所有小于中心点的网格点。请注意，通过这样操作，我们已将搜索的点数减半了。
- 4 求出其余网格点的中心，从第 2 步重复执行，直到在查询的任何一侧剩下一个网格点。这两个点标记含有 X_q 的单元的边界。

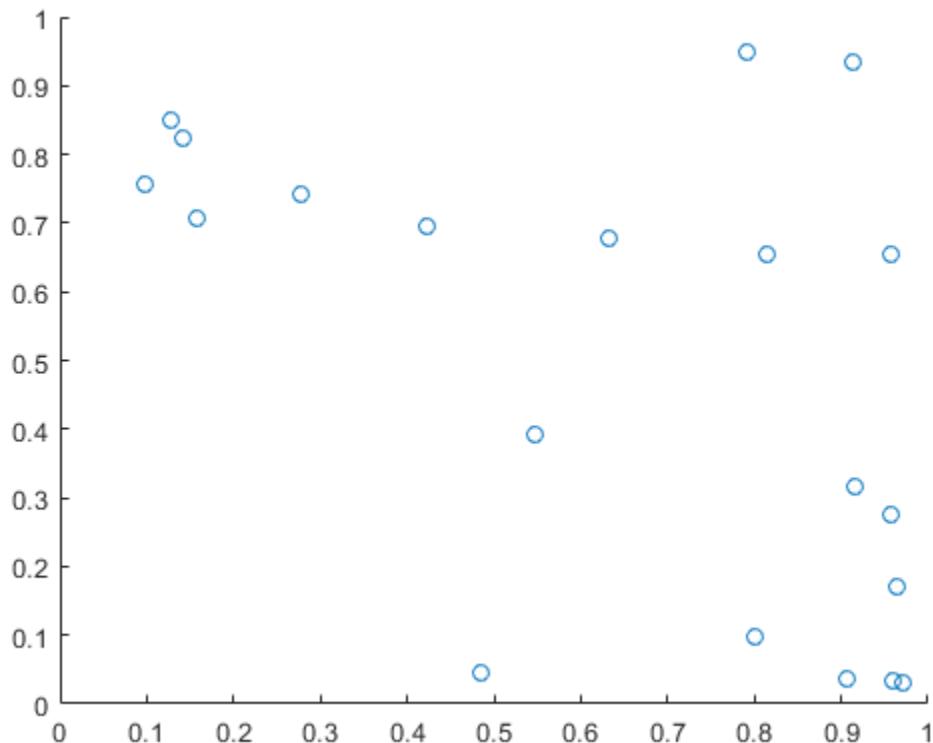


以下示例说明了二分查找的强大功能。在电子化信用卡授权出现前，保护商户免遭信用卡欺诈购买行为的唯一方法是将每个客户的信用卡上的账号与“不良”账号名单进行比较。这类名单是装订好的册子，其中

包含数以万计的按升序排列的卡号。对于一笔交易，要搜索包含 10,000 个账号的名单，需要进行多少次比较？结果表明，对于任意包含 n 个有序项的列表，最大比较次数不超过将列表对分的次数，即 $\log_2(n)$ 。因此，信用卡搜索需要的比较次数不超过 $\log_2(10e3)$ ，即大约 13 次。考虑到执行顺序搜索时所需的比较次数，这是一个非常了不起的结果。

与此相反，假设一个涉及散点数据集的问题。

```
x = rand(20,1);
y = rand(20,1);
scatter(x,y)
```



求查询点邻近的点所需的操作次数要多得多。如果您的数据可以逼近为一个网格，则基于网格的插值可以节省大量计算和内存用量。

如果数据是分散的，则可以使用“内插散点数据”（第 8-37 页）中详细描述的工具。

插值与拟合

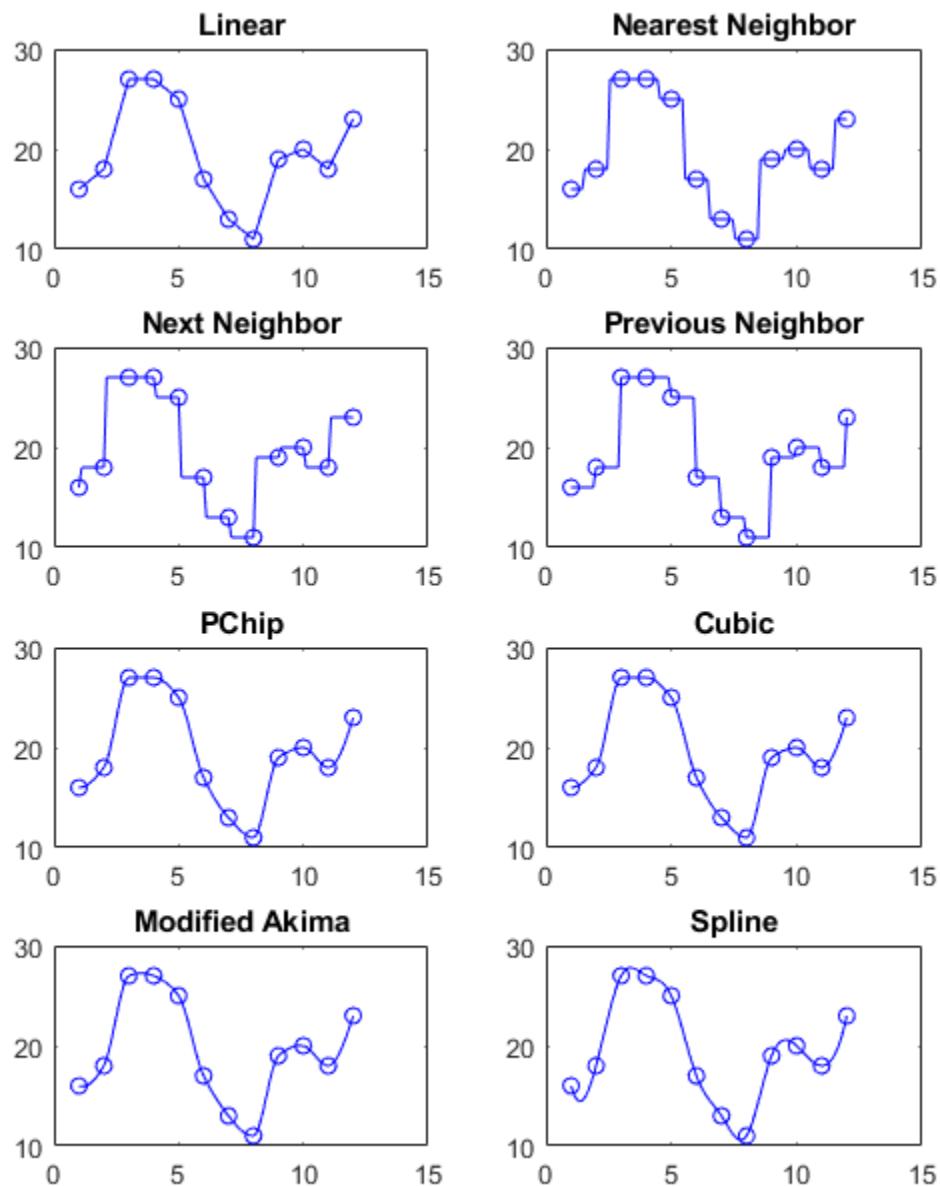
MATLAB 中提供的插值方法可创建经过样本数据点的插值函数。如果要查询一个样本位置的插值函数，就需要取回位于该样本数据点的值。对比插值、曲线和曲面拟合算法则不需要通过样本数据点。

插值方法

基于网格的插值提供多种不同的插值方法。在选择插值方法时，切记有些方法比其他方法需要更多的内存或更长的计算时间。但是，需要权衡这些资源，以实现结果所需要的平滑度。下面的表格提供了每种方法的优点、取舍和要求。

方法	说明	连续性	内存用量和性能	要求
最近邻点	在查询点插入的值是距样本网格点最近的值。	不连续	<ul style="list-style-type: none"> 最低内存要求 最快计算时间 	<ul style="list-style-type: none"> 每个维度需要 2 个网格点。
后邻点	在查询点插入的值是下一个抽样网格点的值。	不连续	其内存要求和计算时间与最近邻点法相同。	<ul style="list-style-type: none"> 仅用于一维插值。 需要至少 2 个网格点。
前邻点	在查询点插入的值是上一个抽样网格点的值。	不连续	其内存要求和计算时间与最近邻点法相同。	<ul style="list-style-type: none"> 仅用于一维插值。 需要至少 2 个网格点。
线性	在查询点插入的值基于各维中邻点网格点处数值的线性插值。	C0	<ul style="list-style-type: none"> 比最近邻点需要更多内存。 比最近邻点需要更多计算时间。 	<ul style="list-style-type: none"> 每个维需要至少 2 个网格点。
Pchip	在查询点插入的值基于邻点网格点处数值的保形分段三次插值。	C1	<ul style="list-style-type: none"> 比线性插值方法需要更多内存。 比线性插值方法需要更长的计算时间。 	<ul style="list-style-type: none"> 仅用于一维插值。 需要至少 4 个网格点。
三次	在查询点插入的值基于各维中邻点网格点处数值的三次插值。	C1	<ul style="list-style-type: none"> 比线性插值方法需要更多内存。 比线性插值方法需要更长的计算时间。 	<ul style="list-style-type: none"> 网格必须有均匀间距，但是每维中的间距不必相同。 每维需要至少 4 个网格点。
修正 Akima	在查询点插入的值基于次数最大为 3 的多项式的分段函数，使用各维中相邻网格点的值进行计算。为防过冲，已修正 Akima 公式。	C1	<ul style="list-style-type: none"> 与样条插值具有相似的内存要求。 比三次插值需要更长的计算时间，但通常少于样条插值的计算时间。 	<ul style="list-style-type: none"> 每个维需要至少 2 个网格点。
样条曲线	在查询点插入的值基于各维中邻点网格点处数值的三次插值。	C2	<ul style="list-style-type: none"> 比三次插值需要更多内存。 比三次插值需要更长的计算时间。 	<ul style="list-style-type: none"> 每维需要 4 个网格点。

此图窗对各种一维数据插值方法进行了直观比较。



插值方法的对比

MATLAB 以多种方式提供与基于网格的插值相关的支持：

- **interp** 系列函数: `interp1`、`interp2`、`interp3` 和 `interpnn`。
- **griddedInterpolant** 类。

interp 系列函数和 **griddedInterpolant** 支持 N 维基于网格的插值。但是，使用 **griddedInterpolant** 类比 **interp** 函数具有内存和性能方面的优势。而且，**griddedInterpolant** 类提供可配合任意维数的网格化数据使用的统一界面。

interp 系列函数的插值

- “**interp1** 函数” (第 8-15 页)
- “以 **interp1** 进行一维外插” (第 8-16 页)
- “**interp2** 函数” (第 8-18 页)
- “**interp3** 函数” (第 8-20 页)
- “**interpn** 函数” (第 8-21 页)

interp1 函数

此示例说明如何使用 **interp1** 函数的 '**pchip**' 方法对一组样本值进行插值。

函数 **interp1** 执行一维插值。其最常见的形式为：

```
Vq = interp1(X,V,Xq,method)
```

其中，**X** 是坐标向量，**V** 向量包含这些坐标处的值。**Xq** 向量包含作为插入位置的查询点，可选的 **method** 方法指定四种插值方法之一：'**nearest**'、'**linear**'、'**pchip**' 或 '**spline**'。

创建一组一维网格点 **X** 和对应的样本值 **V**。

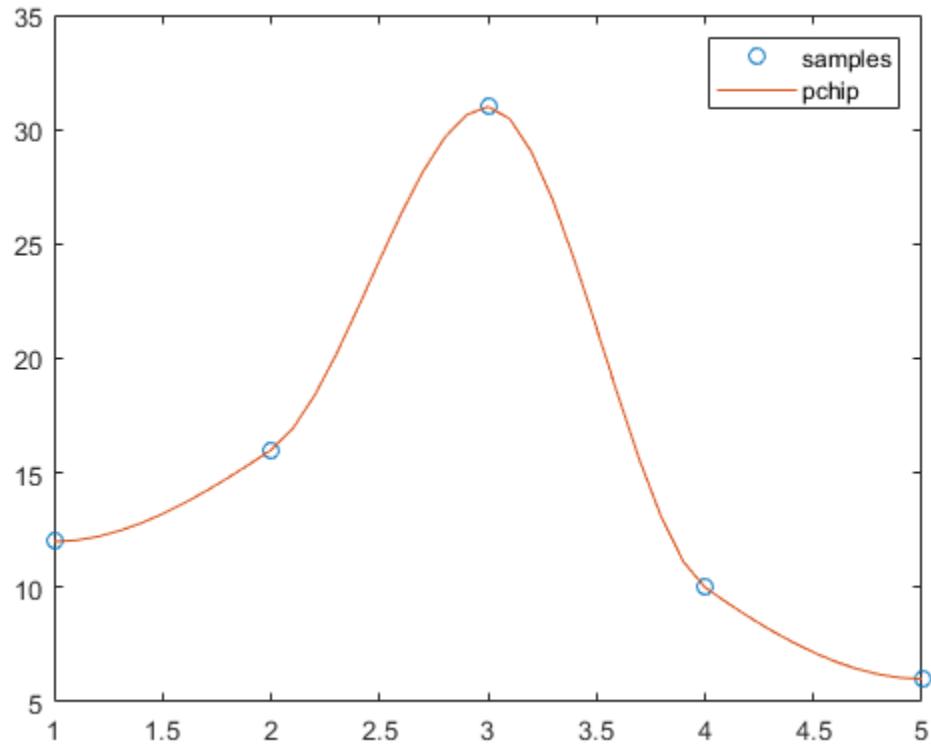
```
X = [1 2 3 4 5];
V = [12 16 31 10 6];
```

以 0.1 为间距在更小的区间上插值。

```
Xq = (1:0.1:5);
Vq = interp1(X,V,Xq,'pchip');
```

绘制样本和插入的值。

```
plot(X,V,'o');
hold on
plot(Xq,Vq,'-');
legend('samples','pchip');
hold off
```



以 `interp1` 进行一维外插

此示例说明如何使用 '`extrap`' 选项在样本点的域之外插值。

定义样本点和值。

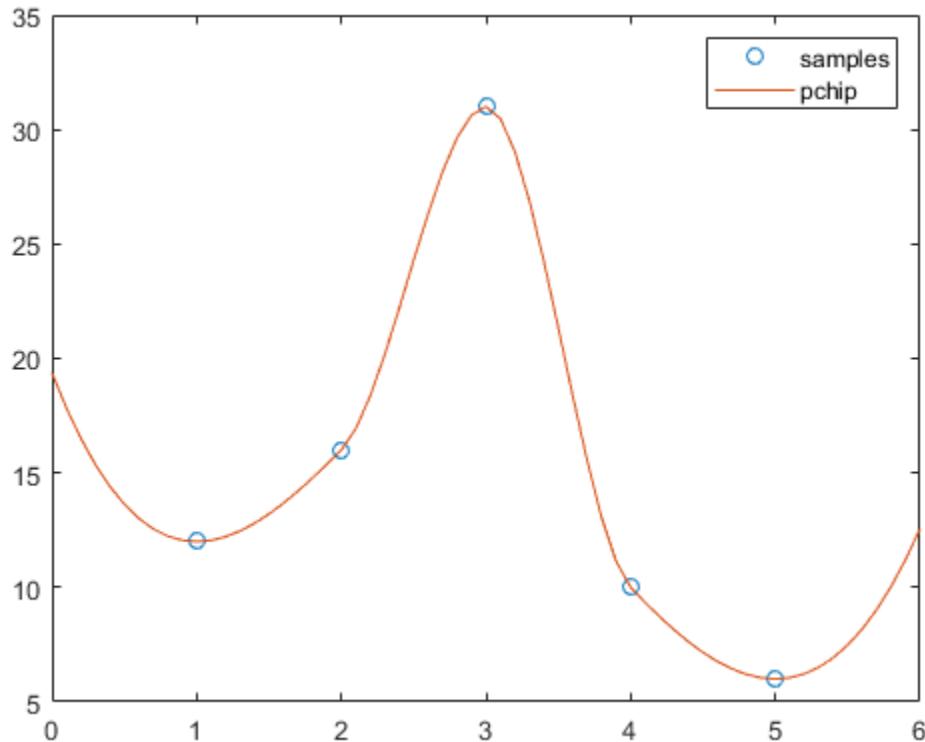
```
X = [1 2 3 4 5];
V = [12 16 31 10 6];
```

指定查询点 `Xq`, 这些查询点延伸到 `X` 的定义域以外。

```
Xq = (0:0.1:6);
Vq = interp1(X,V,Xq,'pchip','extrap');
```

绘制结果。

```
figure
plot(X,V,'o');
hold on
plot(Xq,Vq,'-');
legend('samples','pchip');
hold off
```



在另外的方法中，可以引入更多点，从而更好也控制外插区域中的行为。例如，可通过使用重复的值对域进行延伸，从而对曲线进行约束，使其在外插区域保持扁平。

```
X = [0 1 2 3 4 5 6];
V = [12 12 16 31 10 6 6];
```

指定进一步延伸到 X 的域之外的查询点 Xq。

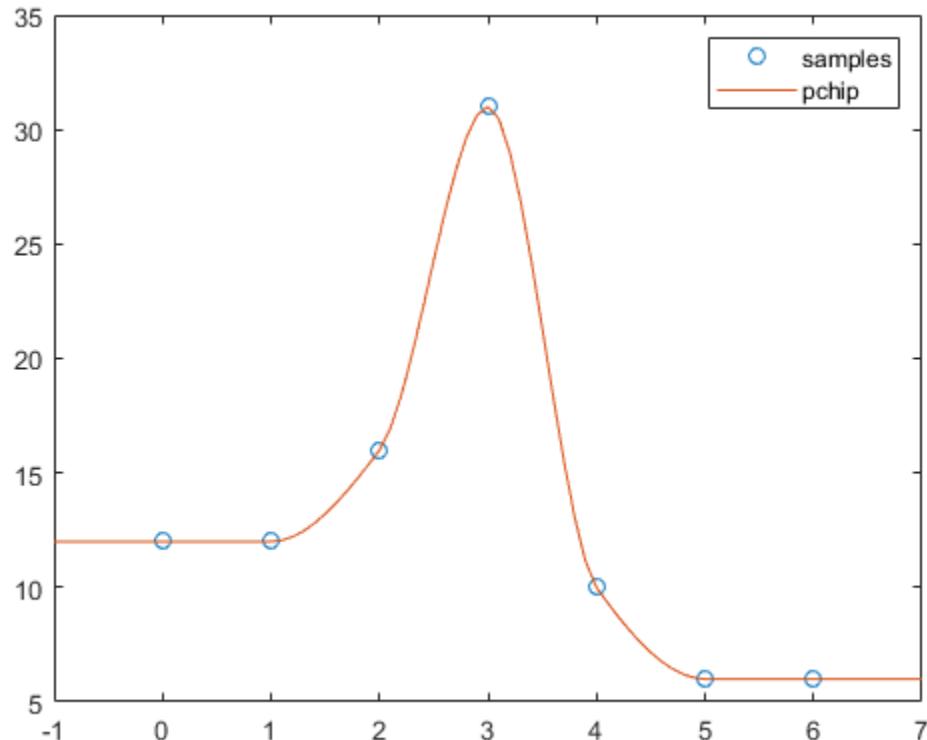
```
Xq = (-1:0.1:7);
```

使用 'pchip' 插值。您可以省略 'extrap' 选项，因为它是 'pchip'、'makima' 和 'spline' 方法的默认选项。

```
Vq = interp1(X,V,Xq,'pchip');
```

绘制结果。

```
figure
plot(X,V,'o');
hold on
plot(Xq,Vq,'-');
legend('samples','pchip');
hold off
```



interp2 函数

此示例说明如何使用 `interp2` 函数在更精细的网格上对粗采样的 `peaks` 函数进行插值。

`interp2` 和 `interp3` 函数分别执行二维和三维插值，并且它们以 `meshgrid` 格式对网格进行插值。
`interp2` 的调用语法具有以下一般形式：

`Vq = interp2(X,Y,V,Xq,Yq,method)`

其中，`X` 和 `Y` 是以 `meshgrid` 格式定义网格的坐标数组，`V` 是包含网格点处的值的数组。`Xq` 和 `Yq` 是包含要插值的查询点坐标位置的数组。可以选择使用 `method` 指定四种插值方法之一：'`'nearest'`'、'`'linear'`'、'`'cubic'`' 或 '`'spline'`'。

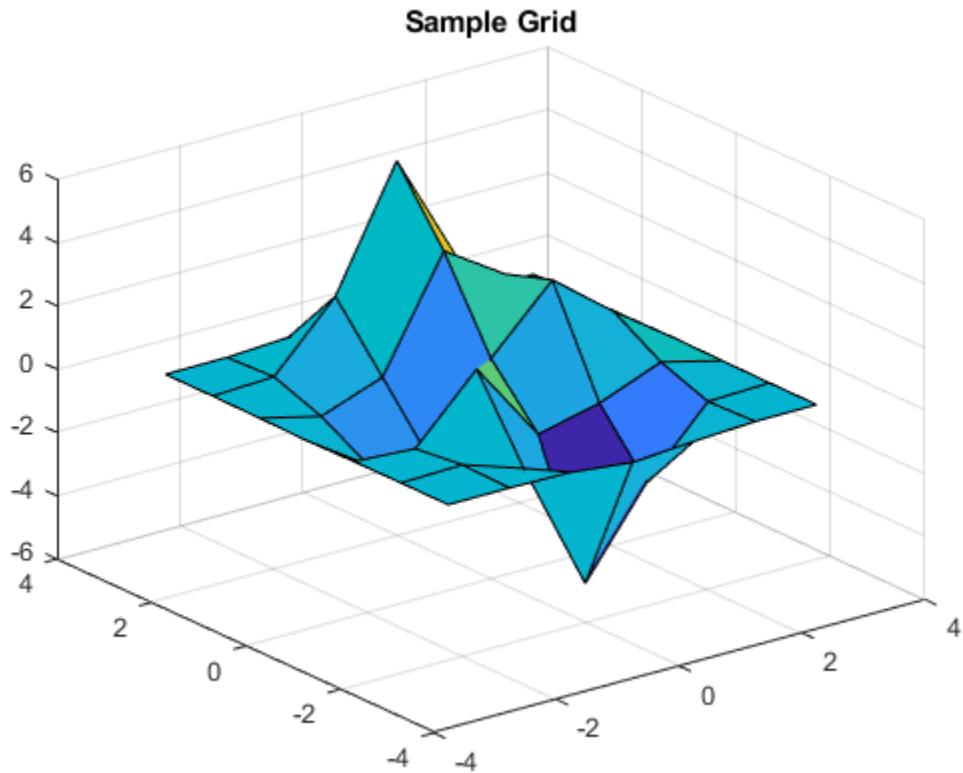
由 `X` 和 `Y` 构成的网格点必须单调递增并且符合 `meshgrid` 格式。

创建粗网格和对应的样本值。

```
[X,Y] = meshgrid(-3:1:3);
V = peaks(X,Y);
```

绘制样本值。

```
surf(X,Y,V)
title('Sample Grid');
```



生成更精细的网格用于插值。

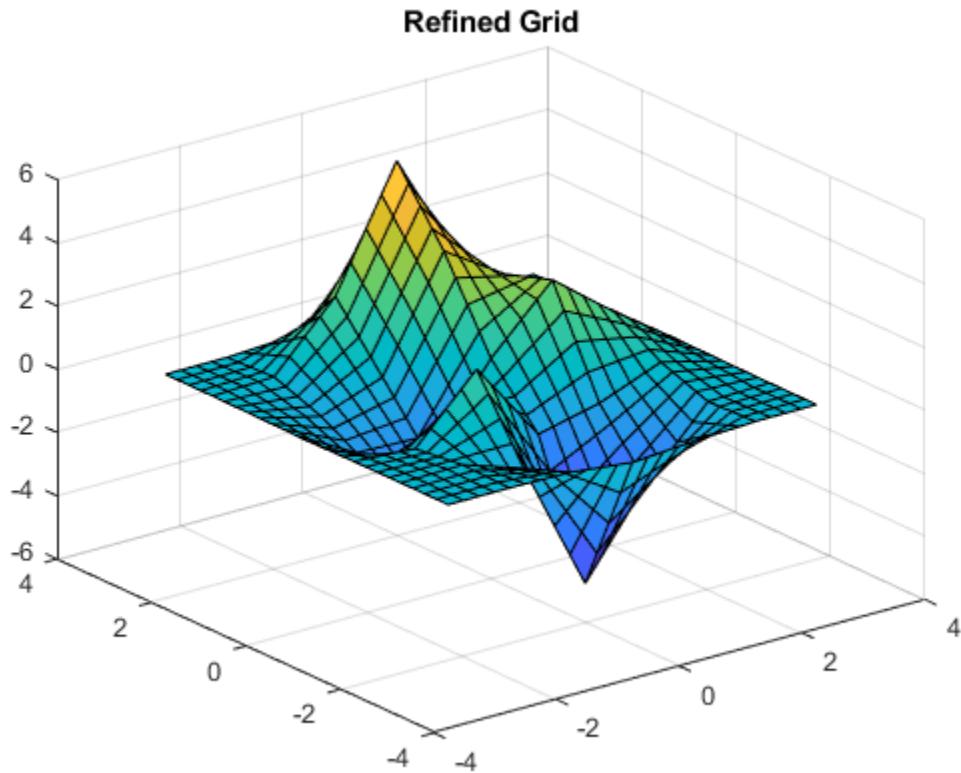
```
[Xq,Yq] = meshgrid(-3:0.25:3);
```

在查询点位置使用 `interp2` 插值。

```
Vq = interp2(X,Y,V,Xq,Yq,'linear');
```

绘制结果。

```
surf(Xq,Yq,Vq);
title('Refined Grid');
```



interp3 函数

此示例说明如何使用 `interp3` 在单个查询点位置对三维函数进行插值，并将其与解析表达式所生成的值进行比较。

`interp3` 的工作方式与 `interp2` 相同，不同的是它采用两个额外参数：一个表示样本网格中的第三个维度，另一个表示查询点中的第三个维度，

`Vq = interp3(X,Y,Z,V,Xq,Yq,Zq,method)`。

与使用 `interp2` 一样，提供给 `interp3` 的网格点必须单调递增，并且符合 `meshgrid` 格式。

定义用于生成 X、Y 和 Z 输入值的函数。

```
generatedvalues = @(X,Y,Z)(X.^2 + Y.^3 + Z.^4);
```

创建样本数据。

```
[X,Y,Z] = meshgrid((-5:25:5));
V = generatedvalues(X,Y,Z);
```

在特定查询点位置插值。

```
Vq = interp3(X,Y,Z,V,2.35,1.76,0.23,'cubic')
```

```
Vq = 10.9765
```

将 `Vq` 与解析表达式所生成的值进行比较。

```
V_actual = generatedvalues(2.35,1.76,0.23)
```

```
V_actual = 10.9771
```

interpн 函数

此示例说明如何使用 `interpн` 函数的 '`cubic`' 方法，在更精细的网格上对粗采样的函数进行插值。

函数 `interpн` 在 `ndgrid` 格式的网格上执行 n 维插值。其最常见的形式为：

```
Vq = interpн(X1,X2,X3,...Xn,V,Y1,Y2,Y3,...,Yn,method)
```

其中，`X1,X2,X3,...,Xn` 是以 `ndgrid` 格式定义网格的坐标数组，`V` 是包含网格点处的值的数组。`Y1,Y2,Y3,...,Yn` 是包含要插值的查询点坐标位置的数组。可以选择使用 `method` 指定四种插值方法之一：'`nearest`'、'`linear`'、'`cubic`' 或 '`spline`'。

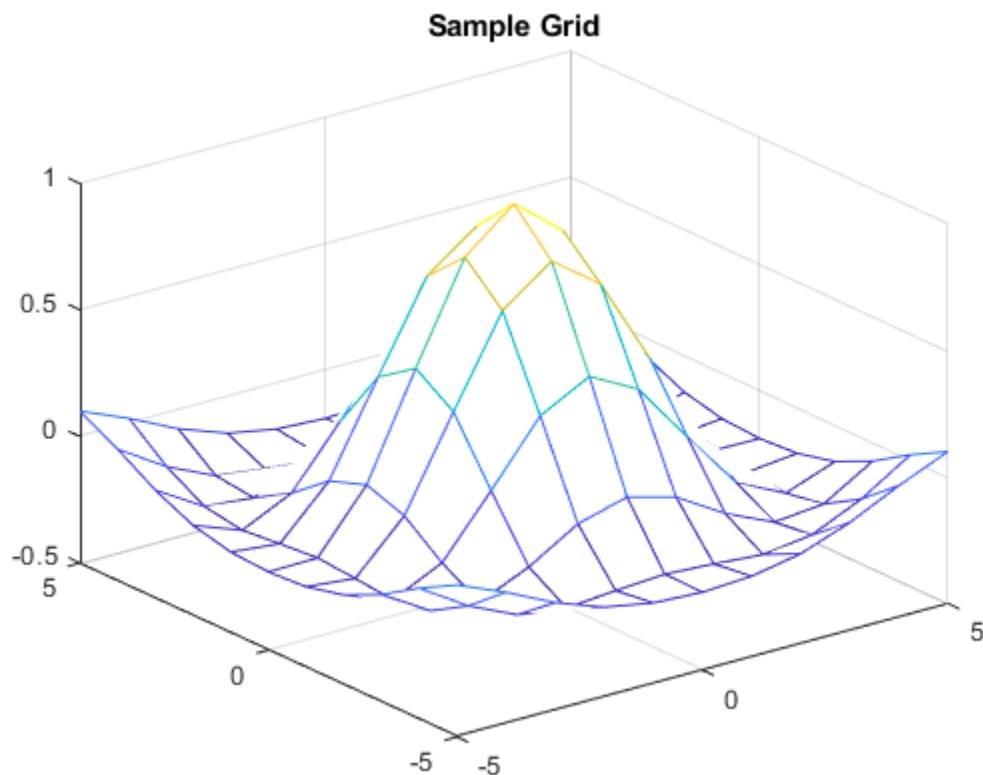
由 `X1,X2,X3,...Xn` 构成的网格点必须单调递增并且符合 `ndgrid` 格式。

创建一组一维网格点和对应的样本值。

```
[X1,X2] = ndgrid((-5:1:5));
R = sqrt(X1.^2 + X2.^2) + eps;
V = sin(R)./(R);
```

绘制样本值。

```
mesh(X1,X2,V)
title('Sample Grid');
```

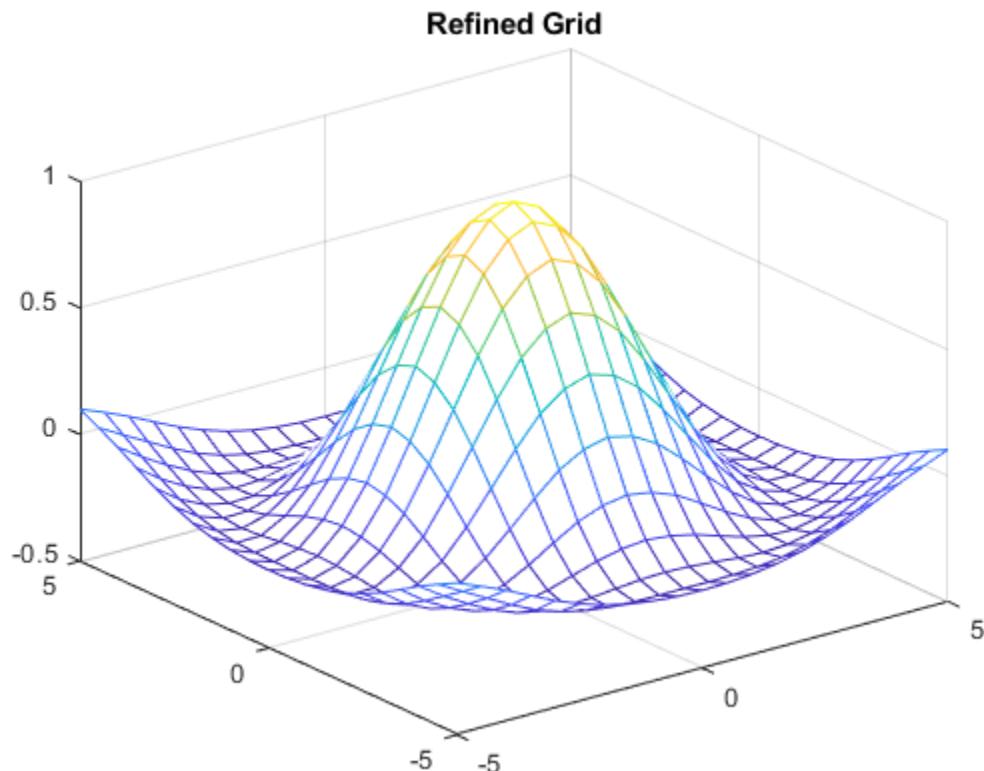


创建更精细的网格用于插值。

```
[Y1,Y2] = ndgrid((-5:5:5));
```

在更精细的网格上插值并绘制结果。

```
Vq = interpn(X1,X2,V,Y1,Y2,'cubic');
mesh(Y1,Y2,Vq);
title('Refined Grid');
```



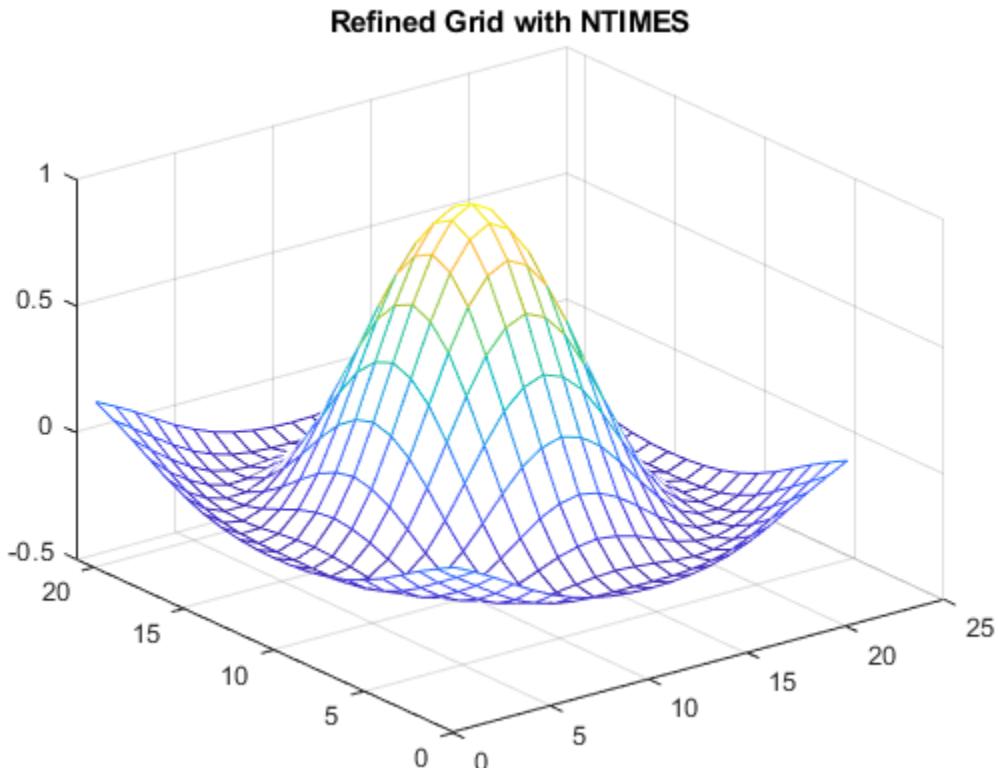
`interpn` 还有另一个语法: `Vq = interpn(V,ntimes,method)`, 利用该语法可以在精度为样本网格整数倍的网格上插值。在上一段代码中, `Y1` 和 `Y2` 查询了每个样本之间包含一个额外点的网格上的插值。以下代码演示了如何使用 `ntimes=1` 获得相同的结果。

使用 `ntimes=1` 在更精细的网格上插值。

```
Vq = interpn(V,1,'cubic');
```

绘制结果。

```
mesh(Vq)
title('Refined Grid with NTIMES');
```



请注意，该绘图的标度与前一示例不同。这是因为我们仅调用了 `mesh` 和传递了值。`mesh` 函数基于 `Vq` 的维度使用默认网格。两种情形下的输出值是相同的。

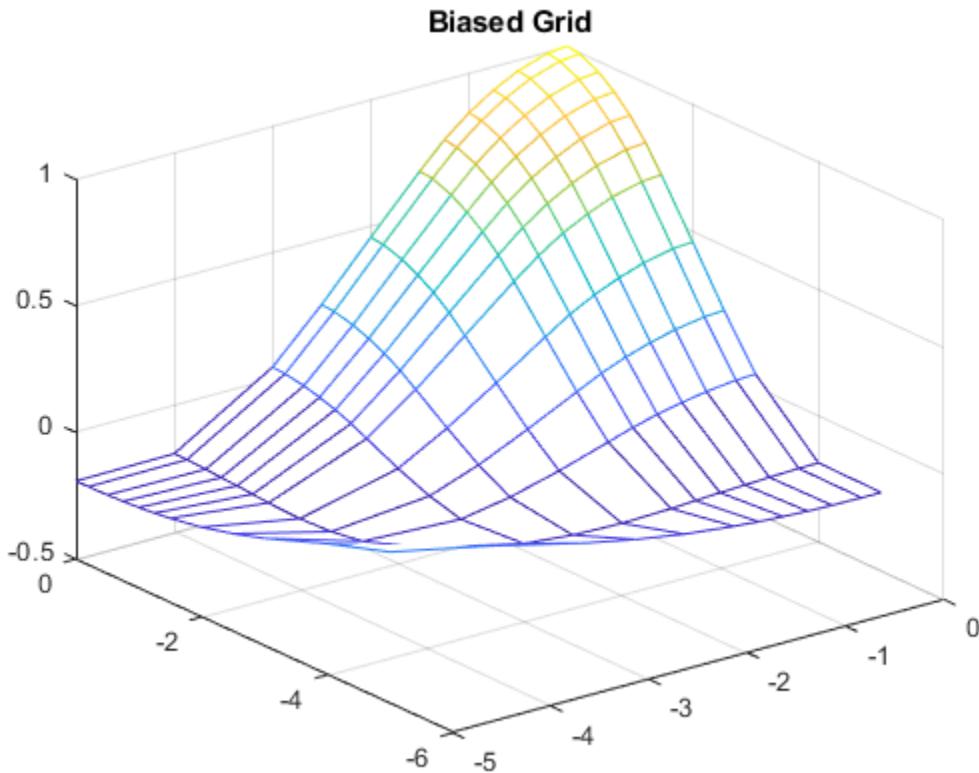
您也可以提供非均匀的查询点网格。如果您有兴趣以更高的密度查询网格区域内的插值，这将非常有用。以下代码说明如何完成此过程。

在偏置网格上插值。

```
[Y1, Y2] = ndgrid([-5 -4 -3 -2.5 -2 -1.5 -1.25 -1 -0.75 -0.5 -0.20 0]);
Vq = interpn(X1,X2,V,Y1,Y2,'cubic');
```

绘制结果。

```
mesh(Y1,Y2,Vq);
title('Biased Grid');
```



用 griddedInterpolant 类插值

与 `interp1` 函数相似，`griddedInterpolant` 类为 n 个维度中基于网格的插值提供单一接口。但是 `griddedInterpolant` 有以下额外优点：

- 这有助于显著提高重复查询插值的性能。
- 它提供了额外的性能改进和节省内存用量，因为将样本点存储为简洁网格（第 8-9 页）。

`griddedInterpolant` 接受符合 `ndgrid` 格式的样本数据。如果希望用 `meshgrid` 数据创建 `griddedInterpolant`，需要将数据转换成 `ndgrid` 格式。有关如何转换二维和三维 `meshgrids` 的演示，请参阅“将 `meshgrid` 数据转换为 `ndgrid` 格式”（第 8-25 页）。

`griddedInterpolant` 类支持下列插值方法（第 8-12 页）（`interp1` 也支持这些方法）：`nearest`、`linear`、`pchip`、`cubic`、`makima` 和 `spline`。但是 `griddedInterpolant` 以更小的开销提供更好的性能。

构建插值

`interpolate` 是用于执行插值的函数。通过调用 `griddedInterpolant` 构造函数和传递样本数据创建插值：网格和对应的样本值。如果希望覆盖默认的“线性”方法，也可以指定插值方法。调用语法具有下面的形式：

- 对于一维插值，可以传递数据集 `x` 和含有对应值的相同长度的向量 `v`。

```
F = griddedInterpolant(x,v)
```

- 对于更高维度，可以提供完整网格。 X_1, X_2, \dots, X_n 将网格指定为一组 n 维数组。这些数组符合 **ndgrid** 格式，大小与样本数组 V 相同。


```
F = griddedInterpolant(X1,X2,...,Xn,V)
```
- 如果已知相邻样本点之间的距离是均匀的，则可以通过只传递样本点 V 让 **griddedInterpolant** 创建一个默认网格。


```
F = griddedInterpolant(V)
```
- 也可以将样本数据的坐标指定为简洁网格。简洁网格由一组向量表示。这些向量然后被置于花括号内封装成元胞数组，例如 $\{x1g, x2g, \dots, xng\}$ 其中向量 $x1g, x2g, \dots, xng$ 定义每一维中的网格坐标。


```
F = griddedInterpolant(\{x1g,x2g,...,xng\},V)
```
- 也可以用任何调用语法将插值方法指定为最终输入参数。本示例指定最近邻点插值。


```
F = griddedInterpolant(\{x1g,x2g,x3g\},V,'nearest')
```

查询插值

griddedInterpolant、 F 求值方式与调用函数相同。您可以将查询点分散或网格化，并按照下面的任何方式将查询点传递给 F ：

- 您可以指定一个 $m \times n$ 矩阵 Xq ，其中包含 n 维的 m 个散点。插入的值 Vq 以一个 $m \times 1$ 向量的形式返回。

$$Vq = F(Xq)$$

- 您还可以将查询点指定为一系列长度为 m 的 n 列向量 $x1q, x2q, \dots, xnq$ 。这些向量表示 n 维的 m 个点。插入的值 Vq 以一个 $m \times 1$ 向量的形式返回。

$$Vq = F(x1q,x2q,...,xnq)$$

- 您可以将查询点指定为一系列表示完整网格的 n 个 n 维数组。数组 $X1q, X2q, \dots, Xnq$ 的大小完全相同，且符合 **ndgrid** 格式。插入的值 Vq 也将大小相同。

$$Vq = F(X1q,X2q,...,Xnq)$$

- 您也可以将查询点指定为简洁网格。 $x1gq, x2gq, \dots, xngq$ 是定义每维中网格点的向量。

$$Vq = F(\{x1gq,x2gq,...,xngq\})$$

例如在二维空间中：

$$Vq = F(\{(0:0.2:10),(-5:0.5:5)\});$$

将 **meshgrid** 数据转换为 **ndgrid** 格式

griddedInterpolant 类接受 **ndgrid** 格式化的样本数据。如果要用 **meshgrid** 数据创建 **griddedInterpolant**，则应首先将其转换为 **ndgrid** 格式。

下面的示例概述将二维 **meshgrid** 数据转换成 **ndgrid** 格式的步骤。通过创建 **meshgrid** 和对应的样本值开始转换：

```
[X,Y] = meshgrid(-2:1:2,-1:1:1);
V=0.75*Y.^3-3*Y-2*X.^2;
```

要将 X 、 Y 和 V 转换成 **ndgrid** 格式，请按下面的步骤操作：

- 1 转置网格中的每个数组以及样本数据。

```
X=X';
Y=Y';
V=V';
```

- 2 现在创建插值。

```
F = griddedInterpolant(X,Y,V);
```

要转换三维 meshgrid，请使用 `permute` 函数。同样地，首先创建 `meshgrid` 和对应的样本值：

```
[X,Y,Z] = meshgrid(-5:5,-3:3,-10:10);
V = X.^3 + Y.^2 + Z;
```

要将 X、Y、Z 和 V 转换成 `ndgrid` 格式，请按下面的步骤操作：

- 1 使用 `permute` 函数互换每个数组的行和列。实际效果是转置每一页。

```
P = [2 1 3];
X = permute(X,P);
Y = permute(Y,P);
Z = permute(Z,P);
V = permute(V,P);
```

- 2 现在创建插值。

```
F = griddedInterpolant(X,Y,Z,V);
```

一维中的 `griddedInterpolant`

本示例显示如何使用 `griddedInterpolant` 及三次插值方法创建和绘制一个一维插值。

创建粗略网格和样本值。

```
X = [1 2 3 4 5];
V = [12 6 15 9 6];
```

使用三次插值方法构建 `griddedInterpolant`。

```
F = griddedInterpolant(X,V,'cubic')
```

```
F =
griddedInterpolant with properties:
```

```
GridVectors: {[1 2 3 4 5]}
Values: [12 6 15 9 6]
Method: 'cubic'
ExtrapolationMethod: 'cubic'
```

`GridVectors` 属性包含指定样本值 V 的坐标的简洁网格。`Method` 属性指定插值方法。请注意，我们在创建 F 时已指定 'cubic'。如果省略 `Method` 参数，则默认插值方法 `linear` 将被赋给 F。

您可以按照访问 F 中的字段的方式来访问 struct 的任意属性。

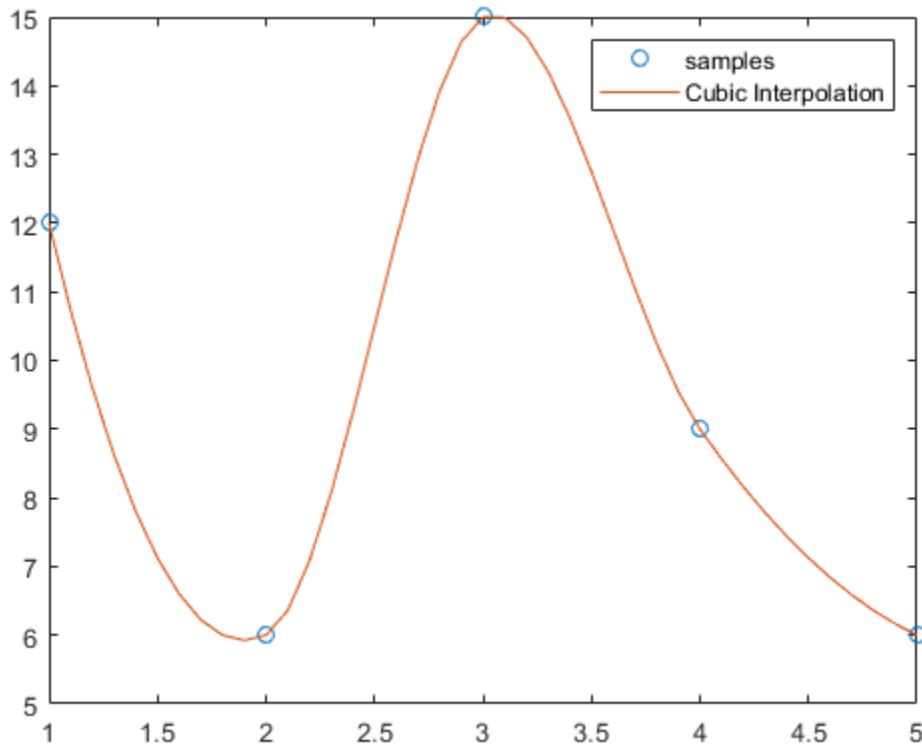
```
F.GridVectors;      % Displays the grid vectors as a cell array
F.Values;          % Displays the sample values
F.Method;          % Displays the interpolation method
```

以 0.1 为间距在更小的区间上插值。

```
Xq = (1:0.1:5);
Vq = F(Xq);
```

绘制结果。

```
plot(X,V,'o');
hold on
plot(Xq,Vq,'-');
legend('samples','Cubic Interpolation');
```



二维中的 griddedInterpolant

本示例显示如何使用 `griddedInterpolant` 来创建和绘制一个二维插值。

在二维和更高维情况下，可以将样本坐标指定为一个 `ndgrid`、简洁网格或默认网格。在此示例中，我们将提供一个 `ndgrid`。

创建粗略网格和样本值。

```
[X,Y] = ndgrid(-1:3:1,-2:3:2);
V = 0.75*Y.^3 - 3*Y - 2*X.^2;
```

构建 `griddedInterpolant`。

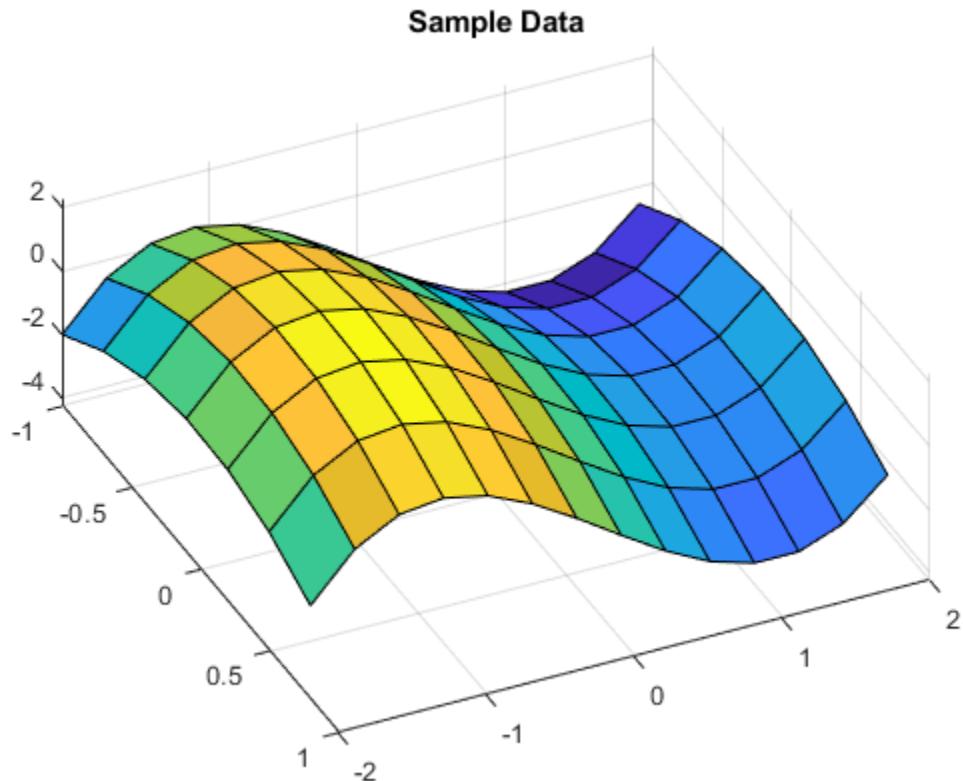
```
F = griddedInterpolant(X,Y,V,'spline');
```

以 0.1 为间距在更小的区间上插值。

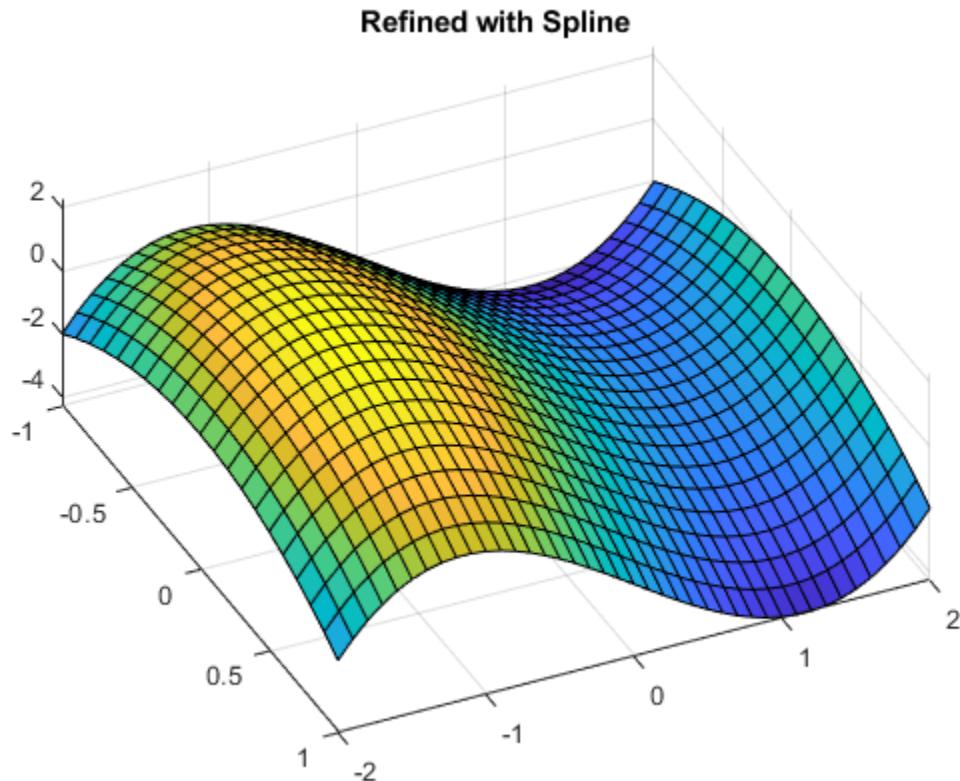
```
[Xq,Yq] = ndgrid(-1:.1:1,-2:.1:2);
Vq = F(Xq,Yq);
```

绘制结果。

```
figure()
surf(X,Y,V)
view(65,60)
title('Sample Data');
```



```
figure()
surf(Xq,Yq,Vq);
view(65,60)
title('Refined with Spline');
```



三维中的 griddedInterpolant

本示例显示如何创建一个三维插值，并在切片平面上求值，以便绘制该平面上的值。

创建完整网格和样本值。

```
[X,Y,Z] = ndgrid((-5:2:5));
V = X.^3 + Y.^2 + Z.^2;
```

构建 griddedInterpolant。

```
F = griddedInterpolant(X,Y,Z,V,'cubic');
```

由于我们创建了完整网格来生成样本值，在将其传递给 griddedInterpolant 时未丢失任何信息。但是在实际情况中，常常将样本数据从磁盘加载到 MATLAB 中。简洁网格在这类情况下非常有利，因为它允许在内存方面节省得多的方式指定整个网格。如果我们已将 V 加载到 MATLAB 中，而没有从完整网格计算，则肯定是在工作区中创建了简洁网格以节省内存。例如，

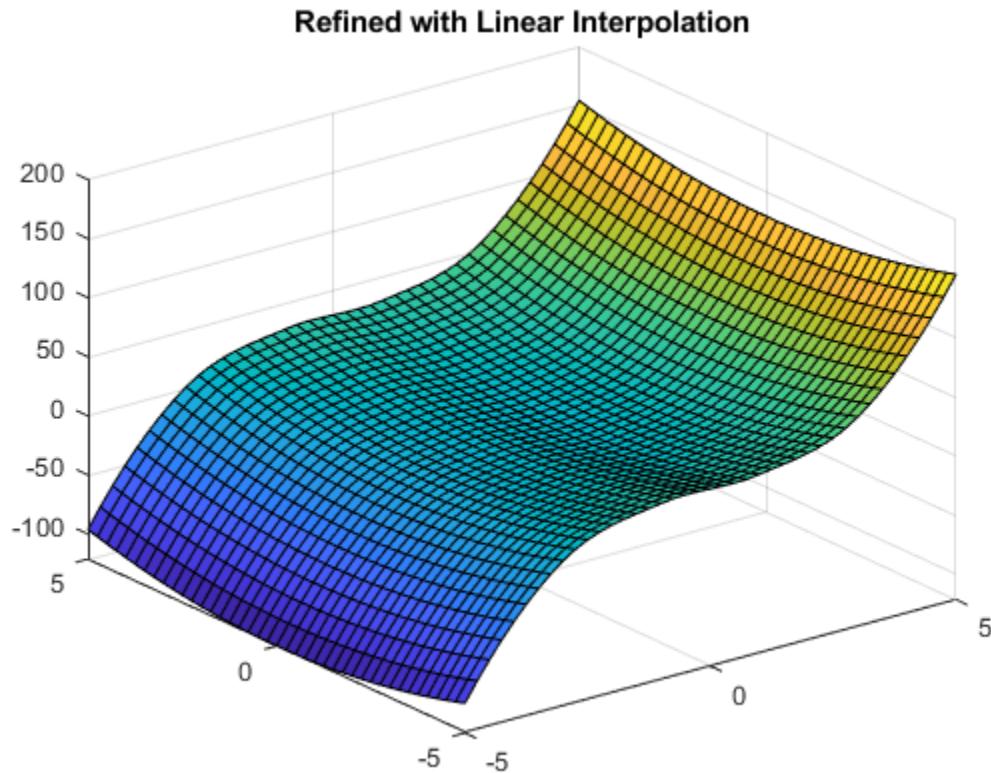
```
gv = {(-5:2:5),(-5:2:5),(-5:2:5)};
F = griddedInterpolant(gv,V,'cubic');
```

现在我们以 0.25 的间距在 $Z = 2$ 处的平面上进行了插值。

```
[Xq,Yq,Zq] = ndgrid((-5:.25:5),(-5:.25:5),2:2);
Vq = F(Xq,Yq,Zq);
```

绘制结果。

```
surf(Xq,Yq,Vq);
title('Refined with Linear Interpolation');
```



四维中的 griddedInterpolant

本示例将会创建一个四维插值并进行单点求值。

- 1 创建粗略网格和样本值。

```
[X1,X2,X3,X4] = ndgrid((-5:2:5));
V = X1.^3 + X2.^2 + X3.^2 + X4;
```

- 2 构建 griddedInterpolant。

```
F = griddedInterpolant(X1,X2,X3,X4,V,'linear');
```

- 3 查询单点处的 griddedInterpolant。

```
Vq = F([-3.2 2.1 4.7 -1.3])
```

MATLAB 输出以下结果：

```
ans =
```

```
-10.1000
```

```
.
```

使用 griddedInterpolant 的其他方式

根据查询点的排列情况，或许某种求值语法较之其他语法更加适合。例如，创建以下插值：

```
[X,Y] = ndgrid(-1:.25:1,-2:.25:2);
V = 0.75*Y.^3-3*Y.^2*X.^2;
F = griddedInterpolant(X,Y,V);
```

使用完整网格来查询 F，以给出网格点处的值：

```
[Xq,Yq] = ndgrid(-1:1:0,-2:1:0);
Vq = F(Xq,Yq);
```

也可以使用简洁网格格式在相同网格上进行插值：

```
gvq = {-1:1:0,-2:1:0};
Vq = F(gvq);
```

或者可以查询单个点：

```
Vq = F(.315,.738)
```

将返回：

```
Vq =
```

-2.1308

或一组随机散点：

```
rng('default')
Vq = F(rand(3,2))
```

将返回：

```
Vq =
```

-3.4919
-3.3557
-0.3515

也可以检查 F 中的 Values：

```
F.Values(1,3)
```

将返回：

```
ans =
```

-0.0313

或者可以替换 Values 数组：

```
F.Values = 2*V;
```

可以动态编辑 F 中的属性。例如，可以按如下方式替换插值方法：

```
F.Method = 'cubic';
```

也可以替换 F 中的 GridVectors。首先，检查 GridVectors：

```
gv = F.GridVectors;
gv{1}
```

gv 是一个元胞数组，gv{1} 显示第一个网格向量：

```
ans =
-1.0000 -0.7500 -0.5000 -0.2500 0 0.2500 0.5000 0.7500 1.0000
```

现在通过创建新的元胞数组 new_gv 替换 F 中的 GridVectors。

```
new_gv = {(0:0.3:1),(0:0.3:1)};
F.GridVectors = new_gv;
```

多个一维值集的插值

此示例说明如何使用 **griddedInterpolant** 一次性插入三个一维数据集。这是比循环插入数据集更快的替代方法。

定义所有值集共用的 x 坐标。

```
x = (1:5)';
```

沿矩阵 V 的列定义样本点集。

```
V = [x, 2*x, 3*x]
```

$V = 5 \times 3$

1	2	3
2	4	6
3	6	9
4	8	12
5	10	15

创建由样本点构成的二维网格。

```
samplePoints = {x, 1:size(V,2)};
```

此简洁表示法指定完整的二维网格。第一个元素 $\text{samplePoints}\{1\}$ 包含 V 的 x 坐标， $\text{samplePoints}\{2\}$ 包含 y 坐标。每个坐标向量的方向无关紧要。

通过将样本点和样本值传递给 **griddedInterpolant** 来创建插值 F 。

```
F = griddedInterpolant(samplePoints,V);
```

在 V 的所有列上沿 x 创建一个间距为 0.5 的二维查询网格。

```
queryPoints = {(1:0.5:5),1:size(V,2)};
```

在每个值集的 x 坐标位置计算该插值。

```
Vq = F(queryPoints)
```

$Vq = 9 \times 3$

1.0000	2.0000	3.0000
1.5000	3.0000	4.5000
2.0000	4.0000	6.0000
2.5000	5.0000	7.5000
3.0000	6.0000	9.0000
3.5000	7.0000	10.5000
4.0000	8.0000	12.0000
4.5000	9.0000	13.5000
5.0000	10.0000	15.0000

另请参阅

griddedInterpolant

相关示例

- “用 griddedInterpolant 类插值” (第 8-24 页)

将二维选择项插入三维网格中

本示例显示如何降低三维的网格平面数组的维度以求解二维插值问题。

在某些应用领域，可能需要插入网格的低维平面；例如，插入三维网格的一个平面。从三维网格中提取网格平面时，生成的数组可能为三维格式。您可以使用 `squeeze` 函数降低网格平面数组在二维中求解问题。

创建三维样本网格和对应的值。

```
[X,Y,Z] = ndgrid(1:5);
V = X.^2 + Y.^2 + Z;
```

从网格中选择二维样本。在此例中，为样本的第三列。

```
x = X(:,3,:);
z = Z(:,3,:);
v = V(:,3,:);
```

二维平面出现在 $Y=3$ 处，因此 Y 维度已固定。 x 、 z 和 v 是 $5 \times 1 \times 5$ 的数组。必须先将数组降低为二维数组才能计算该插值。

使用 `squeeze` 函数将 x 、 z 和 v 向下降低为二维数组。

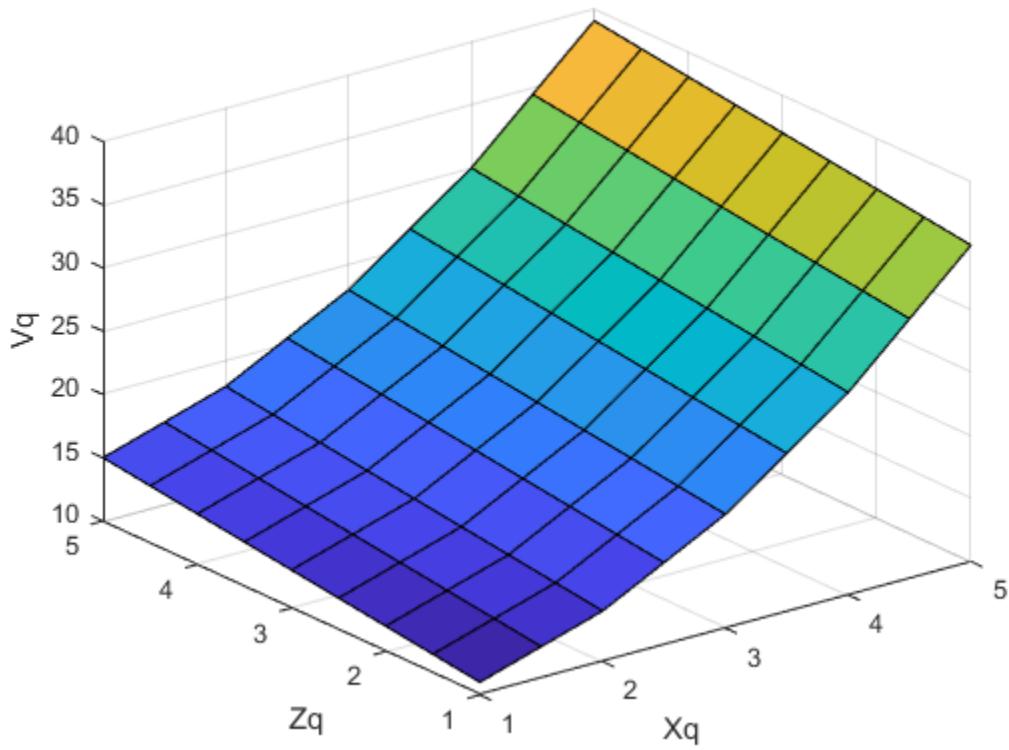
```
x = squeeze(x);
z = squeeze(z);
v = squeeze(v);
```

在更精细的查询点网格上插入二维切片。

```
[Xq,Zq] = ndgrid(1:0.5:5);
Vq = interpn(x,z,v,Xq,Zq);
```

绘制结果。

```
figure
surf(Xq,Zq,Vq);
xlabel('Xq');
ylabel('Zq');
zlabel('Vq');
```



另请参阅

`interp1 | squeeze`

详细信息

- “网格数据表示” (第 8-3 页)

内插散点数据

本节内容

- “散点数据”（第 8-37 页）
- “使用 griddata 和 griddatan 插入散点数据”（第 8-39 页）
- “scatteredInterpolant 类”（第 8-42 页）
- “使用 scatteredInterpolant 类插入散点数据”（第 8-43 页）
- “复数散点数据的插值”（第 8-49 页）
- “解决散点数据插值中的问题”（第 8-51 页）

散点数据

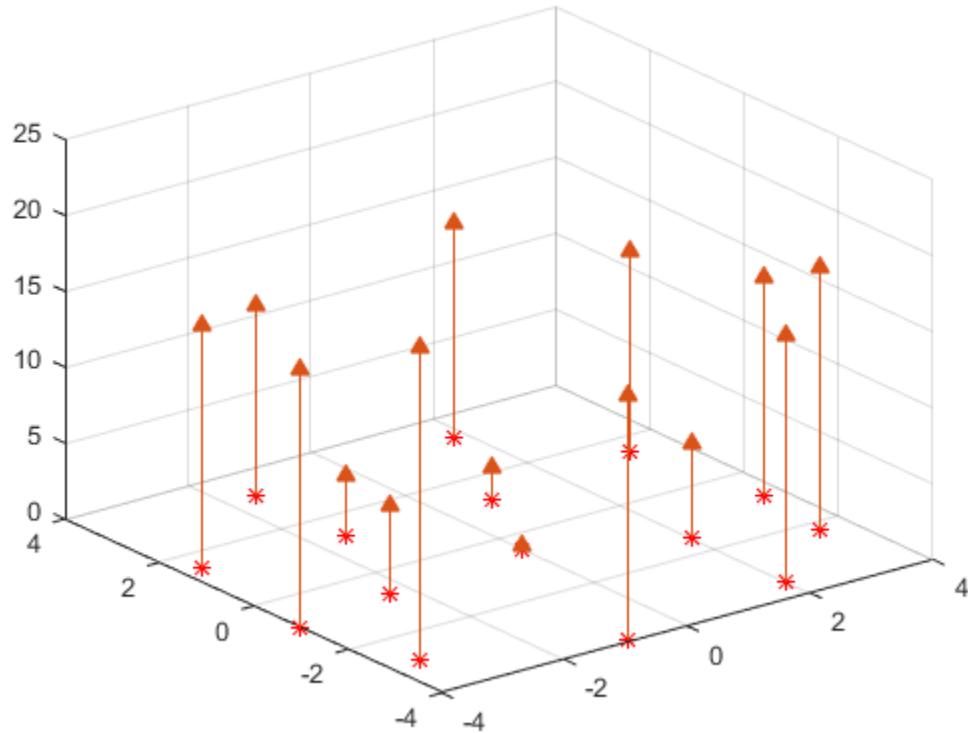
散点数据包含点集 X 和对应值 V ，其中的点没有结构或其相对位置间的顺序。进行散点数据插值有多种方法。一种广泛使用的方法是使用点的 Delaunay 三角剖分。

此示例说明如何通过对点进行三角剖分并按模 V 将顶点提升至与 X 正交的维度，来构建插值曲面。

应用此方法有多种方式。在本示例中，插值分为几个单独的步骤；通常情况下，整个插值过程通过一次函数调用完成。

在抛物面上创建散点数据集。

```
X = [-1.5 3.2; 1.8 3.3; -3.7 1.5; -1.5 1.3; ...
    0.8 1.2; 3.3 1.5; -4.0 -1.0; -2.3 -0.7;
    0 -0.5; 2.0 -1.5; 3.7 -0.8; -3.5 -2.9; ...
    -0.9 -3.9; 2.0 -3.5; 3.5 -2.25];
V = X(:,1).^2 + X(:,2).^2;
hold on
plot3(X(:,1),X(:,2),zeros(15,1), '*r')
axis([-4, 4, -4, 4, 0, 25]);
grid
stem3(X(:,1),X(:,2),V,'^','fill')
hold off
view(322.5, 30);
```



创建 Delaunay 三角剖分，提升顶点，并在查询点 X_q 位置计算插值。

```

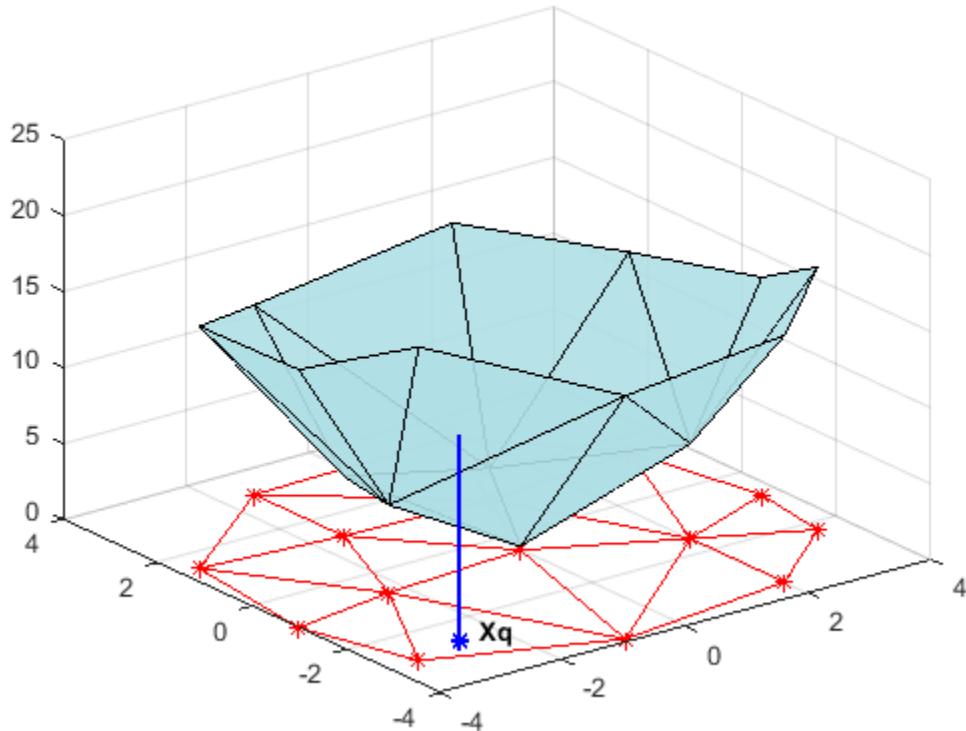
figure('Color','white')
t = delaunay(X(:,1),X(:,2));
hold on

trimesh(t,X(:,1),X(:,2), zeros(15,1), ...
    'EdgeColor','r', 'FaceColor','none')
defaultFaceColor = [0.6875 0.8750 0.8984];
trisurf(t,X(:,1),X(:,2), V, 'FaceColor',...
    defaultFaceColor, 'FaceAlpha',0.9);
plot3(X(:,1),X(:,2),zeros(15,1), '*r')
axis([-4, 4, -4, 4, 0, 25]);
grid
plot3(-2.6,-2.6,0,'*b','LineWidth', 1.6)
plot3([-2.6 -2.6],[-2.6 -2.6],[0 13.52],'-b','LineWidth',1.6)
hold off

view(322.5, 30);

text(-2.0, -2.6, 'Xq', 'FontWeight', 'bold',...
    'HorizontalAlignment','center', 'BackgroundColor', 'none');

```



此步骤通常涉及遍历三角剖分数据结构体以求包围查询点的三角形。找到该点之后，计算值的后续步骤取决于插值方法。您可以计算相邻的最近的点，并使用该点的值（最近邻插值方法）。也可以计算包围三角形的三个顶点值的加权和（线性插值方法）。有关散点数据插值的文本和参考资料中介绍了这些方法及其变化形式。

尽管这里主要说明二维插值，但此方法也可应用于更高的维度。更一般的描述是，给定点集 X 和对应的值 V ，您可以构造 $V = F(X)$ 形式的插值。您可以在查询点 X_q 计算插值，即 $V_q = F(X_q)$ 。这是一个单值函数；对于 X 的凸包中的任何查询点 X_q ，它将产生唯一的值 V_q 。为了产生满意的插值，假定样本数据适用此属性。

MATLAB 提供两种方式执行基于三角剖分的散点数据插值：

- 函数 `griddata` 和 `griddatan`
- `scatteredInterpolant` 类

`griddata` 函数支持二维散点数据插值。`griddatan` 函数支持 N 维散点数据插值；但是对于高于六维的中到大型点集并不实际，因为基本三角剖分需要的内存呈指数增长。

`scatteredInterpolant` 类支持二维和三维空间中的散点数据插值。鼓励使用这种类，因为它的效率更高，容易适应更广泛的插值问题。

使用 `griddata` 和 `griddatan` 插入散点数据

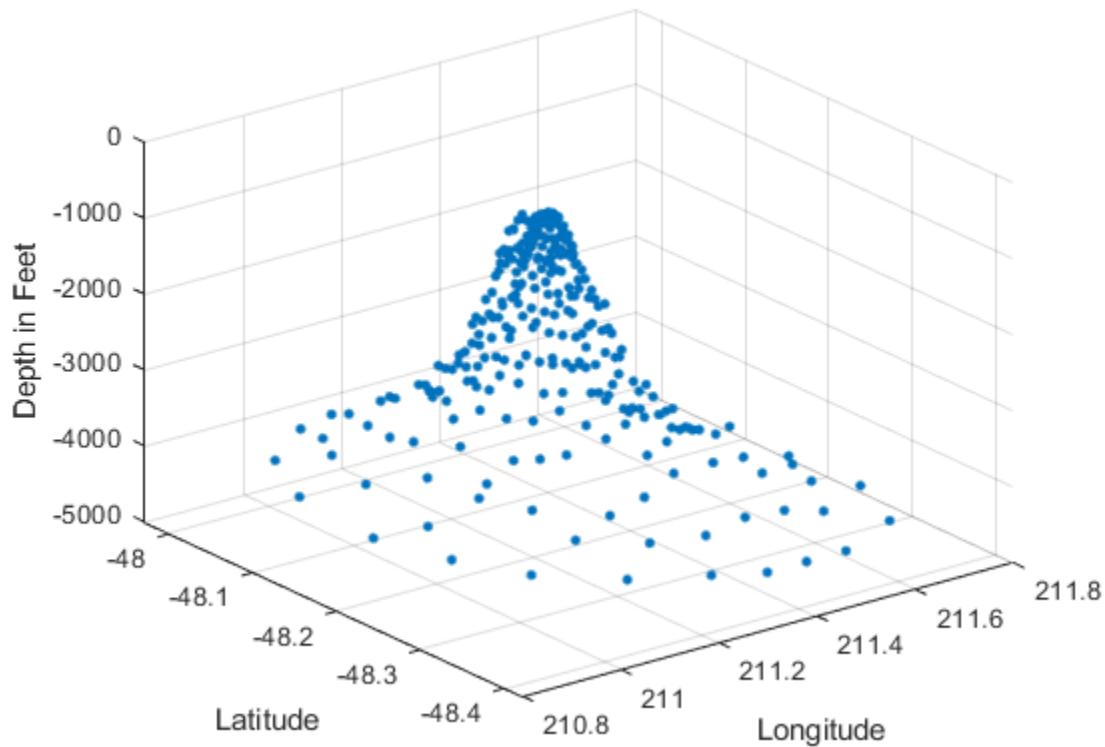
`griddata` 和 `griddatan` 函数接受一组样本点 X 、对应值 V 和查询点 X_q ，然后返回插入的值 V_q 。这两个函数的调用语法类似，主要区别在于二维/三维 `griddata` 函数可让您依据 $X, Y / X, Y, Z$ 坐标来定义

点。这两个函数在预定义的网格点位置插入散点数据，意图是产生网格数据，因此得名。在实际中插值的使用更具一般性。也许需要在点的凸包内的任意位置进行查询。

此示例说明 `griddata` 函数如何在一系列网格点进行散点数据插值，并使用此网格数据创建等高线图。

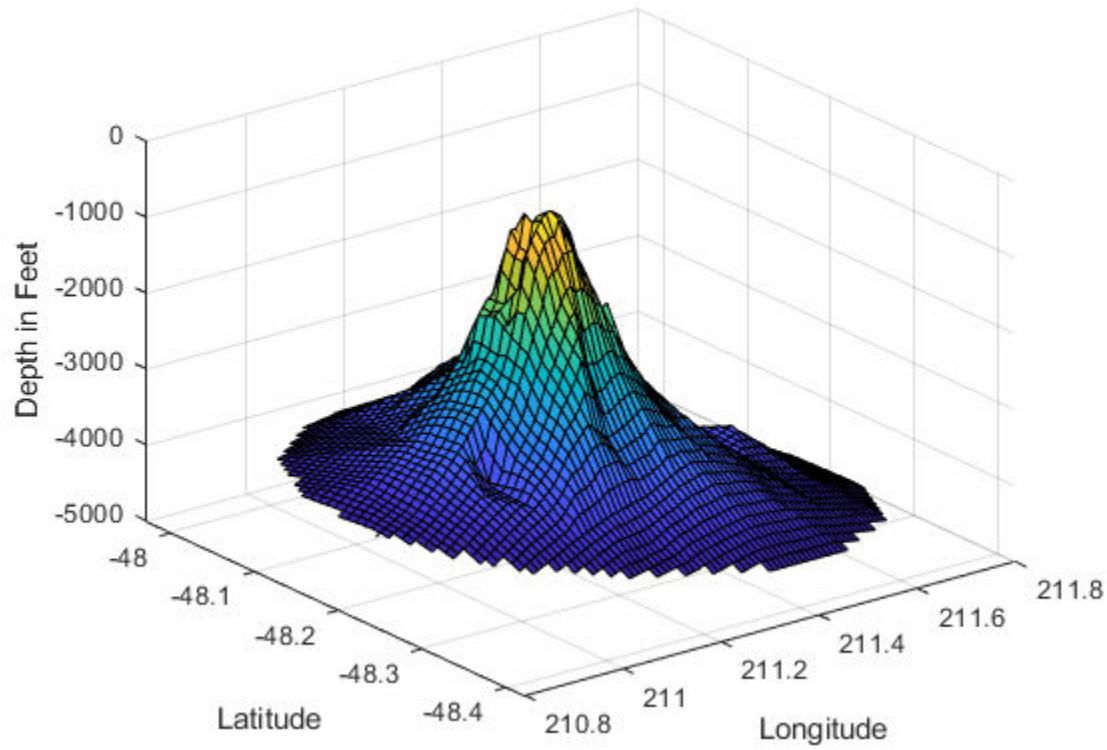
绘制 `seamount` 数据集（海底山是指水下山脉）。该数据集包含一组经度 (x) 和纬度 (y) 位置，以及在这些坐标位置测量的对应 `seamount` 海拔 (z)。

```
load seamount
plot3(x,y,z,'markerSize',12)
xlabel('Longitude')
ylabel('Latitude')
zlabel('Depth in Feet')
grid on
```



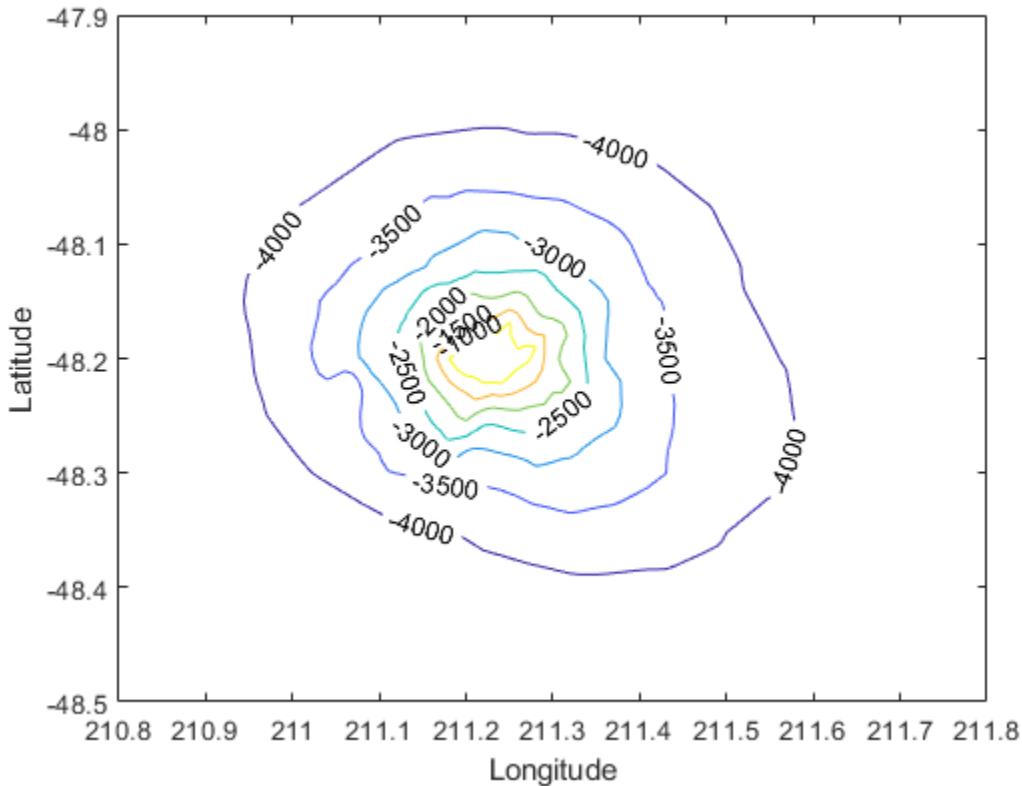
使用 `meshgrid` 在经度-纬度平面创建一组二维网格点，然后使用 `griddata` 在这些点插入对应的深度。

```
figure
[xi,yi] = meshgrid(210.8:0.01:211.8, -48.5:0.01:-47.9);
zi = griddata(x,y,z,xi,yi);
surf(xi,yi,zi);
xlabel('Longitude')
ylabel('Latitude')
zlabel('Depth in Feet')
```



现在，数据已转换为网格化格式，计算并绘制等高线。

```
figure  
[c,h] = contour(xi,yi,zi);  
clabel(c,h);  
xlabel('Longitude')  
ylabel('Latitude')
```



也可以使用 **griddata** 在数据集的凸包中任意位置处进行插值。例如，坐标 (211.3, -48.2) 处的深度由以下值给出：

```
zi = griddata(x,y,z, 211.3, -48.2);
```

每次调用 **griddata** 函数时都会计算基本三角剖分。如果用不同的查询点对相同的数据集重复进行插值，这样可能会影响性能。“使用 **scatteredInterpolant** 类插入散点数据”（第 8-43 页）中描述的 **scatteredInterpolant** 类在这方面效率更高。

MATLAB 软件还提供了 **griddatan** 以支持更高维的插值。调用语法类似于 **griddata**。

scatteredInterpolant 类

在需要通过插值对一组预定义网格点位置求值时，**griddata** 函数可派上用场。实际上，插值问题往往更普遍，**scatteredInterpolant** 类的灵活性更强。这个类具有以下优点：

- 它引入了可高效查询的插值函数。即基本三角剖分只创建一次，然后在后续查询中重复使用。
- 可改变插值方法，与三角剖分无关。
- 数据点处的值也可更改，与三角剖分无关。
- 数据点可被逐步添加到现有插值中，无需触发整个重算过程。相对于样本点总数而言，如果要编辑的点数较少，那么数据点的删除和移动效率很高。
- 对于凸包外部的点，提供了外插功能以求得近似值。有关详细信息，请参阅“外插散点数据”（第 8-62 页）。

scatteredInterpolant 提供以下插值方法：

- 'nearest' - 最近邻点插值，其中的插值曲面是不连续的。
- 'linear' - 线性插值（默认值），其中的插值曲面是 C0 连续的。
- 'natural' - 自然邻点插值，其中的插值曲面是 C1 连续的，但样本点除外。

`scatteredInterpolant` 类支持二维和三维空间中的散点数据插值。可以通过调用 `scatteredInterpolant`，传递插值点位置和对应值，并使用内插和外插方法作为可选参数，来创建插值。有关可用于创建和计算 `scatteredInterpolant` 的语法的详细信息，请参阅 `scatteredInterpolant` 参考页。

使用 `scatteredInterpolant` 类插入散点数据

本示例显示如何使用 `scatteredInterpolant` 插入 `peaks` 函数的散点样本。

创建散点数据集。

```
X = -3 + 6.*gallery('uniformdata',[250 2],0);
V = peaks(X(:,1),X(:,2));
```

创建插值。

```
F = scatteredInterpolant(X,V)
```

```
F =
scatteredInterpolant with properties:
```

```
    Points: [250x2 double]
    Values: [250x1 double]
    Method: 'linear'
    ExtrapolationMethod: 'linear'
```

Points 属性表示数据点的坐标，**Values** 属性表示关联值。**Method** 属性表示执行插值的插值方法。**ExtrapolationMethod** 属性表示当查询点位于凸包外部时使用的外插方法。

按照访问 `struct` 的字段的相同方式访问 `F` 的属性。例如，使用 `F.Points` 检查数据点的坐标。

求出插值。

`scatteredInterpolant` 可以按下标方式求某点的插值。其求值方式与函数相同。可以按如下方式求出插入的值。这种情况下，位于查询位置的值由 `Vq` 给出。可以对单个查询点求值：

```
Vq = F([1.5 1.25])
```

```
Vq = 1.3966
```

也可以传递单个坐标：

```
Vq = F(1.5, 1.25)
```

```
Vq = 1.3966
```

可以在点位置向量处求值：

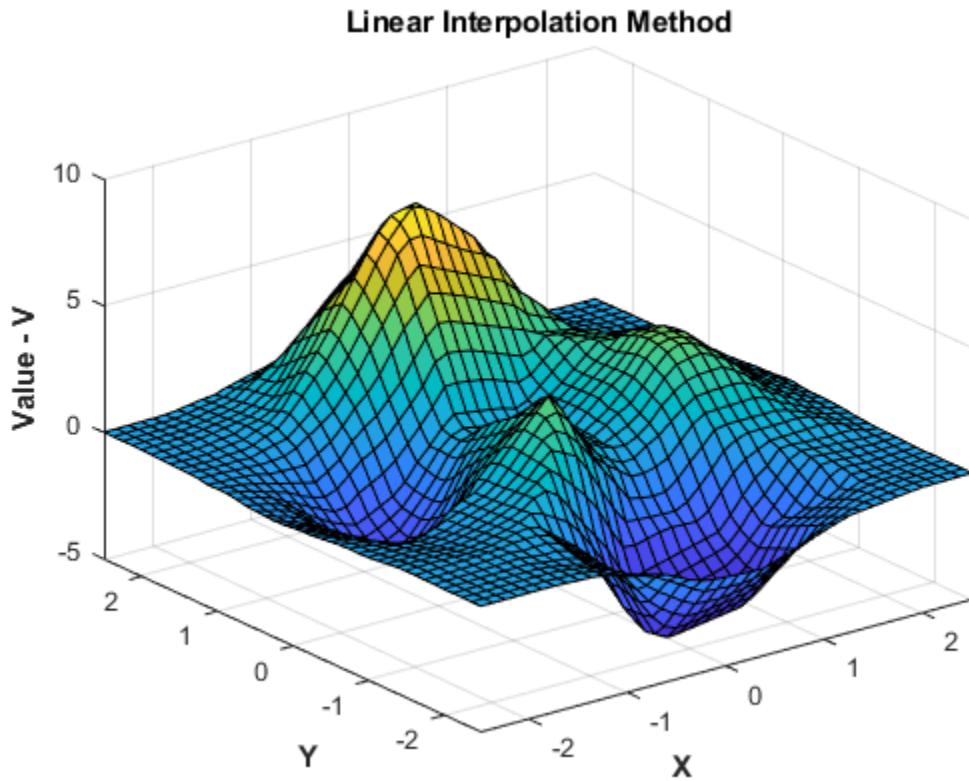
```
Xq = [0.5 0.25; 0.75 0.35; 1.25 0.85];
Vq = F(Xq)
```

```
Vq = 3×1
```

```
1.0880
1.8127
2.3472
```

您可以计算网格点位置的 F 并绘制结果。

```
[Xq,Yq] = meshgrid(-2.5:0.125:2.5);
Vq = F(Xq,Yq);
surf(Xq,Yq,Vq);
xlabel('X','fontweight','b'), ylabel('Y','fontweight','b');
zlabel('Value - V','fontweight','b');
title('Linear Interpolation Method','fontweight','b');
```



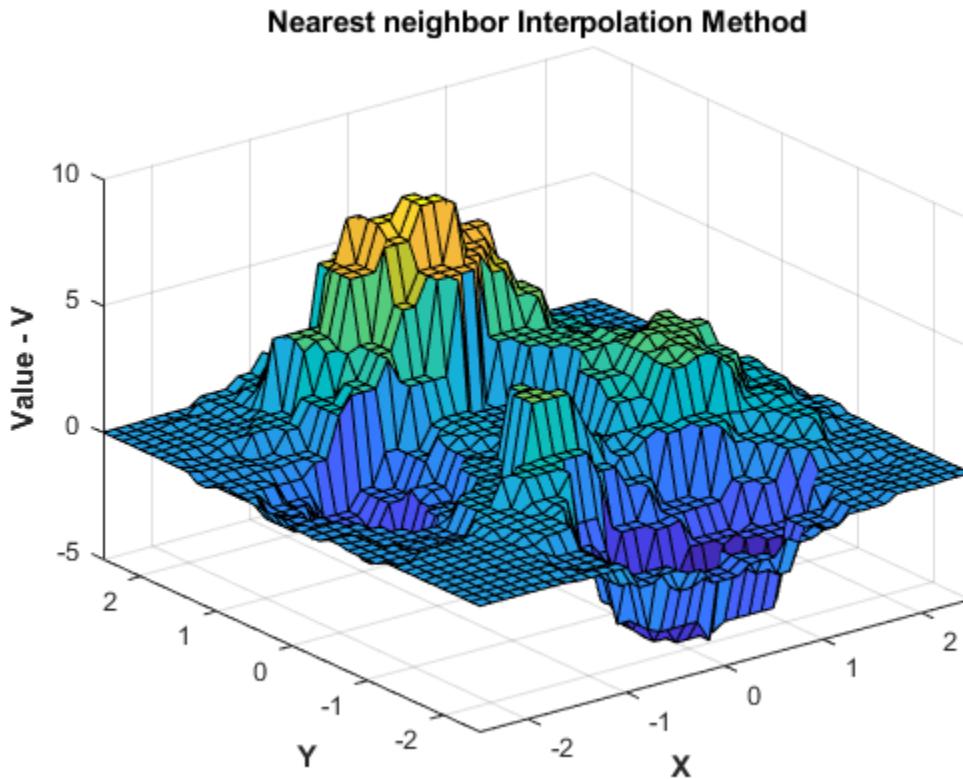
更改插值方法。

可以动态更改插值方法。将方法设置为 'nearest'。

```
F.Method = 'nearest';
```

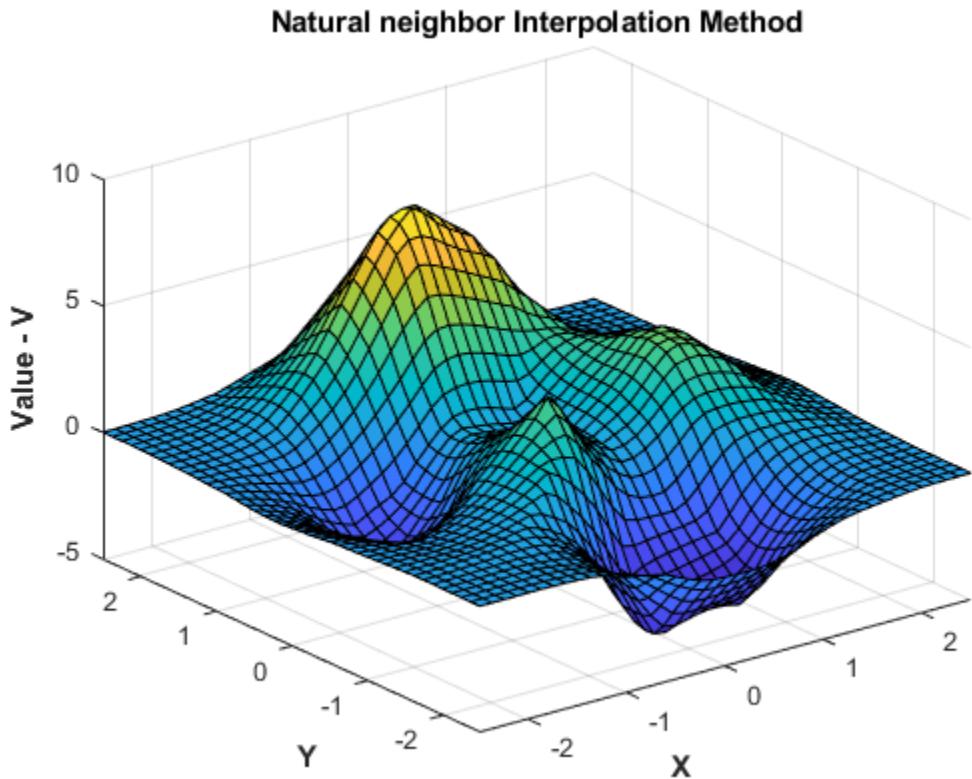
按以前方式重新计算并绘制插值。

```
Vq = F(Xq,Yq);
figure
surf(Xq,Yq,Vq);
xlabel('X','fontweight','b'), ylabel('Y','fontweight','b')
zlabel('Value - V','fontweight','b')
title('Nearest neighbor Interpolation Method','fontweight','b');
```



将插值方法更改为自然邻点，重新计算并绘制结果。

```
F.Method = 'natural';
Vq = F(Xq,Yq);
figure
surf(Xq,Yq,Vq);
xlabel('X','fontweight','b'),ylabel('Y','fontweight','b')
zlabel('Value - V','fontweight','b')
title('Natural neighbor Interpolation Method','fontweight','b');
```



替换样本数据位置处的值。

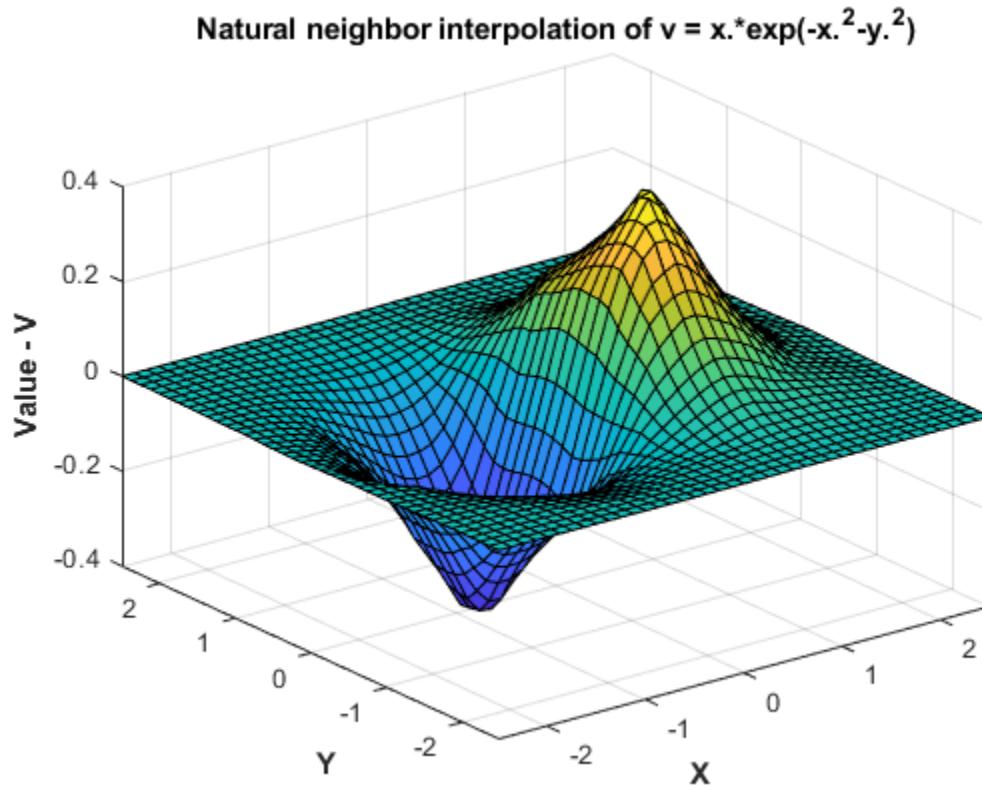
可以动态更改位于样本数据位置 X 处的值 V。这在实际操作中很有用，因为一些插值问题可能在同一位置处具有多组值。例如，假设要为一个由位置 (x, y, z) 和对应的分量速度向量 (V_x, V_y, V_z) 定义的三维速度场进行插值。通过依次将每个速度分量赋给值属性 (V)，可以进行插值。此方法具有重要的性能优点，因为它允许重复使用相同的插值，而不会引起每次计算新值的开销。

以下步骤说明如何更改示例中的值。您将使用表达式 $v = xe^{-x^2 - y^2}$ 来计算值。

```
V = X(:,1).*exp(-X(:,1).^2-X(:,2).^2);
F.Values = V;
```

求出插值并绘出结果。

```
Vq = F(Xq,Yq);
figure
surf(Xq,Yq,Vq);
xlabel('X','fontweight','b'), ylabel('Y','fontweight','b')
zlabel('Value - V','fontweight','b')
title('Natural neighbor interpolation of v = x.*exp(-x.^2-y.^2)')
```



向现有插值添加其他点位置和值。

与使用扩充数据集完全重新计算相比，此方法可执行有效的更新。

在添加样本数据时，同时添加点位置和对应值很重要。

沿用该示例，按如下方式创建新样本点：

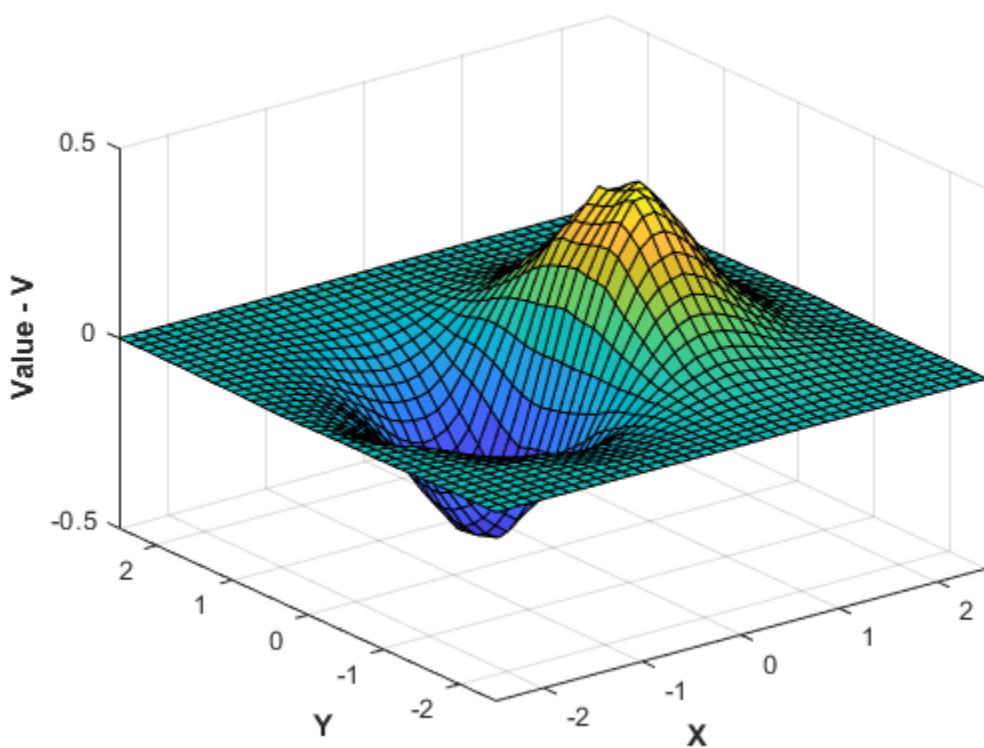
```
X = -1.5 + 3.*rand(100,2);
V = X(:,1).*exp(-X(:,1).^2-X(:,2).^2);
```

给三角剖分添加新点和对应值。

```
F.Points(end+(1:100),:) = X;
F.Values(end+(1:100)) = V;
```

求出改进的插值并绘出结果。

```
Vq = F(Xq,Yq);
figure
surf(Xq,Yq,Vq);
xlabel('X','fontweight','b'), ylabel('Y','fontweight','b');
zlabel('Value - V','fontweight','b');
```



从插值中移除数据。

可以从插值逐步移除样本数据点。也可以从插值移除数据点和对应值。这对移除虚假离群值很有用。

在移除样本数据时，同时移除点位置和对应值很重要。

移除第 25 个点。

```
F.Points(25,:) = [];
F.Values(25) = [];
```

移除第 5 至 15 个点。

```
F.Points(5:15,:) = [];
F.Values(5:15) = [];
```

保留 150 个点，移除其余点。

```
F.Points(150:end,:) = [];
F.Values(150:end) = [];
```

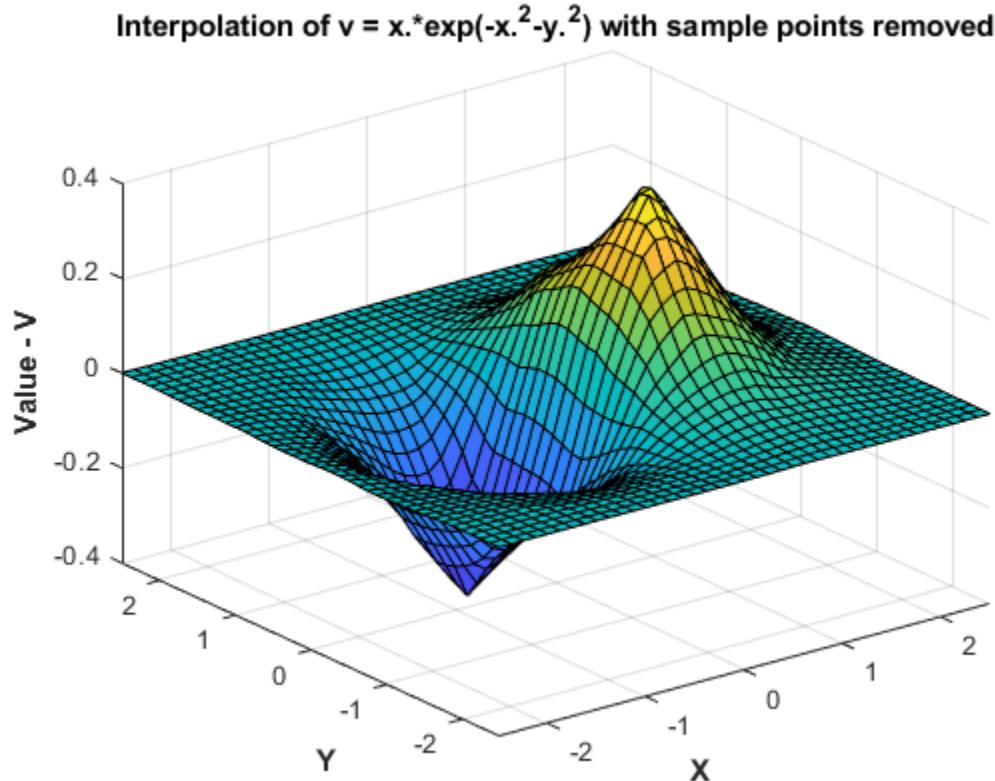
在求值和绘图时，这将创建一个粗略的曲面图：

```
Vq = F(Xq,Yq);
figure
surf(Xq,Yq,Vq);
xlabel('X','fontweight','b'), ylabel('Y','fontweight','b');
```

```

zlabel('Value - V','fontweight','b');
title('Interpolation of v = x.*exp(-x.^2-y.^2) with sample points removed')

```



复数散点数据的插值

此示例说明如何在每个样本位置的值是复数时插入散点数据。

创建样本数据。

```

X = -3 + 6 .* gallery('uniformdata',[250 2],0);
V = complex(X(:,1).*X(:,2), X(:,1).^2 + X(:,2).^2);

```

创建插值。

```
F = scatteredInterpolant(X,V);
```

创建网格查询点，求出网格点处的插值。

```
[Xq,Yq] = meshgrid(-2.5:0.125:2.5);
Vq = F(Xq,Yq);
```

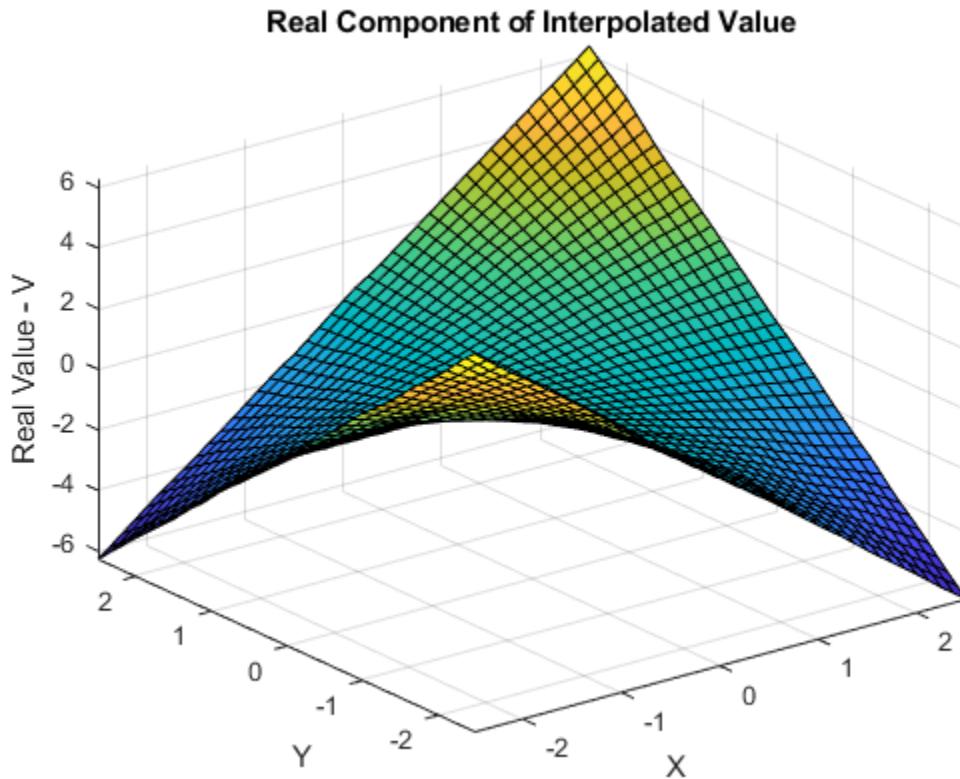
绘出 Vq 的实部。

```

VqReal = real(Vq);
figure
surf(Xq,Yq,VqReal);
xlabel('X');

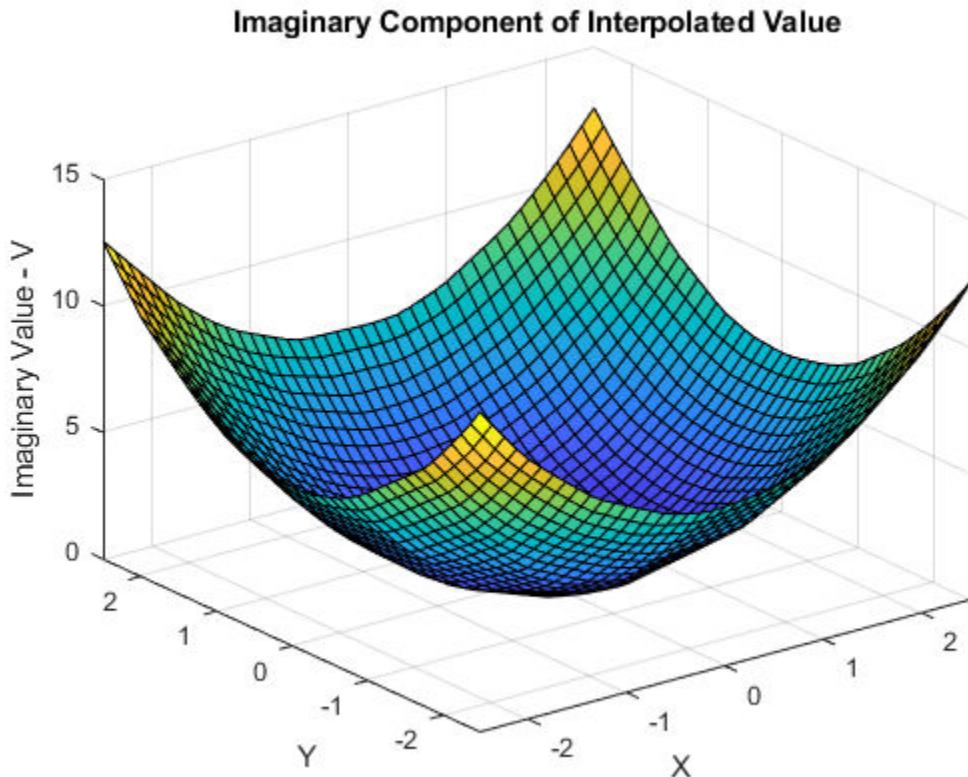
```

```
ylabel('Y');
zlabel('Real Value - V');
title('Real Component of Interpolated Value');
```



绘出 V_q 的虚部。

```
VqImag = imag(Vq);
figure
surf(Xq,Yq,VqImag);
xlabel('X');
ylabel('Y');
zlabel('Imaginary Value - V');
title('Imaginary Component of Interpolated Value');
```



解决散点数据插值中的问题

前面部分中说明的许多示例处理的是在平滑曲面上采样的点集的插值。而且这些点的间距相对均匀。例如，点簇之间距离相对来说不大。此外，在点位置的凸包中插值求值情况良好。

在处理现实世界的插值问题时，数据可能更有挑战性。数据可能是从测量设备获取的，这些设备的读数可能不准确或者给出的是离群值。基础数据的变化可能不平滑，数值可能从点到点急剧跳跃。本节为您识别并解决散点数据插值的问题提供一些指导。

输入含有 NaN 的数据

应预处理含有 NaN 值的样本数据，以移除 NaN 值，因为此数据对插值没有作用。如果移除 NaN 后，还应移除对应的数据值/坐标，以确保一致。如果样本数据中存在 NaN 值，则在调用构建函数时会出错。

下面的示例说明如何移除 NaN。

创建一些数据，然后用 NaN 替换一些项：

```
x = rand(25,1)*4-2;
y = rand(25,1)*4-2;
V = x.^2 + y.^2;

x(5) = NaN; x(10) = NaN; y(12) = NaN; V(14) = NaN;
```

这段代码出错：

```
F = scatteredInterpolant(x,y,V);
```

正确的做法是，找出 NaN 的样本点索引后再构造插值：

```
nan_flags = isnan(x) | isnan(y) | isnan(V);
```

```
x(nan_flags) = [];
y(nan_flags) = [];
V(nan_flags) = [];
```

```
F = scatteredInterpolant(x,y,V);
```

如果点位置属于矩阵形式，则下面的示例是相似的。首先，创建数据，用 NaN 替换一些项。

```
X = rand(25,2)*4-2;
V = X(:,1).^2 + X(:,2).^2;
```

```
X(5,1) = NaN; X(10,1) = NaN; X(12,2) = NaN; V(14) = NaN;
```

这段代码出错：

```
F = scatteredInterpolant(X,V);
```

找出 NaN 的样本点索引，然后构造插值：

```
nan_flags = isnan(X(:,1)) | isnan(X(:,2)) | isnan(V);
```

```
X(nan_flags,:) = [];
V(nan_flags) = [];
```

```
F = scatteredInterpolant(X,V);
```

插值输出 NaN 值

当使用 'linear' 或 'natural' 方法查询凸包外部的点时，`griddata` 和 `griddatan` 返回 NaN 值。但是，如果使用 'nearest' 方法查询相同的点，可能会得到数值结果。这是因为凸包内部和外部都存在查询点的最近邻点。

如果要计算凸包外部的近似值，应使用 `scatteredInterpolant`。有关详细信息，请参阅“外插散点数据”（第 8-62 页）。

重复点位置的处理

输入数据很少“十全十美”，应用程序不得不处理重复的数据点位置。数据集中位于相同位置的两个或多个数据点可能有不同的对应值。这种情况下，`scatteredInterpolant` 合并这些点，计算对应值的平均值。此示例说明 `scatteredInterpolant` 如何在具有重复点的数据集上执行插值。

1 创建位于平面上的一些样本数据：

```
x = rand(100,1)*6-3;
y = rand(100,1)*6-3;
```

```
V = x + y;
```

2 通过将点 50 的坐标赋给点 100 产生一个重复的点位置：

```
x(50) = x(100);
y(50) = y(100);
```

3 创建插值。请注意，F 含有 99 个唯一数据点：

F = scatteredInterpolant(x,y,V)
4 检查与第 50 个点相关联的值：

F.Values(50)

这个值是第 50 和第 100 个原始值的平均值，因为这两个数据点位于同一位置：

(V(50)+V(100))/2

这种情形下，插值以合理的方式解决模棱两可的情况。但是在某些实例中，数据点可能很靠近而不是完全一致，这些位置中的值可能不同。

在有些插值问题中，多组样本值可能对应于同一位置。例如，可在不同时间段记录同一位置的一组值。为提高效率，可以插入一组读数，然后替换数值，以插入下一组读数。

在存在重复点位置的情况下替换值时，始终使用一致的数据管理。假定有两组值与 100 个数据点位置相关联，希望通过替换数值依次插入每一组。

1 考虑这两组值：

V1 = x + 4*y;
V2 = 3*x + 5*y

2 创建插值。scatteredInterpolant 合并重复位置，插值含有 99 个唯一采样点：

F = scatteredInterpolant(x,y,V1)

通过 **F.Values = V2** 直接替换数值意味着将 100 个值赋给 99 个样本。由于前面合并操作的上下文已经丢失，样本位置的数量将与样本值的数量不符。消除这种不一致后才能对插值进行查询。

在这种更复杂的情形中，需要移除重复项后才创建和编辑插值。使用 **unique** 函数求出唯一点的索引。**unique** 也可以输出标识重复点索引的参数。

```
[~, I, ~] = unique([x y],'first','rows');
I = sort(I);
x = x(I);
y = y(I);
V1 = V1(I);
V2 = V2(I);
F = scatteredInterpolant(x,y,V1)
```

现在可以使用 **F** 对第一个数据集进行插值。然后替换数值，对第二个数据集进行插值。

F.Values = V2;

在编辑 scatteredInterpolant 时提高效率

scatteredInterpolant 允许编辑表示样本值的属性 (**F.Values**) 和表示插值方法的属性 (**F.Method**)。由于这些属性与基本三角剖分无关，因此可以高效率地执行编辑。但是，与使用大型数组一样，在编辑数据时应当小心谨慎，以免意外创建不必要的副本。若多个变量引用了同一个数组，然后该数组又被编辑过，在这种情况下就会产生副本。

在以下情况不会产生副本：

```
A1 = magic(4)
A1(4,4) = 11
```

但是下面这种情况就会产生副本，因为该数组已被另一个变量引用。在进行编辑后，数组 **A1** 和 **A2** 就无法再共享同一个数据：

```
A1 = magic(4)
A2 = A1
A1(4,4) = 32
```

同样，如果将数组传递给函数，然后在该函数中编辑此数组，则可能会发生深拷贝，具体取决于数据的管理方式。`scatteredInterpolant` 含有数据，其表现类似 MATLAB 语言中的数组，称为值对象。当应用程序按照文件中的函数这种方式组织时，MATLAB 语言能发挥最佳性能。在命令行进行原型构建可能达不到同样的性能水平。

下面的示例说明这种行为，但是应当注意到，本例中性能的提高不会推广到 MATLAB 中的其他函数。

这段代码不会产生最佳性能：

```
x = rand(1000000,1)*4-2;
y = rand(1000000,1)*4-2;
z = x.*exp(-x.^2-y.^2);
tic; F = scatteredInterpolant(x,y,z); toc
tic; F.Values = 2*z; toc
```

可以将代码放入函数文件中，以更加高效率地执行。

如果程序是由文件中的函数组成的，那么 MATLAB 可以获得代码执行的整体情况；这就使 MATLAB 可以优化性能。在命令行上键入代码时，MATLAB 无法预料在下一步将键入什么内容，因此无法实现相同的优化水平。通过创建可重用的函数开发应用程序是通用惯例，并且建议这样做，MATLAB 将优化此设置中的性能。

凸包附近的插值结果效果不佳

Delaunay 三角剖分非常适合散点数据插值问题，因为它有优越的几何属性可产生良好的结果。这些属性是：

- 拒绝条形三角形/四面体，支持更多等边等面形体。
- 空外接圆属性，隐含地定义了点之间的最近邻关系。

空外接圆属性确保插入的值受查询位置的邻点中样本点的影响。尽管有这些特性，在某些情形中数据点的分布可能导致结果不良，在样本数据集的凸包附近通常发生这种情况。当插值产生意外结果时，通过绘制样本数据和基本三角剖分图常常能够深入了解问题。

本示例显示内插表面在边界附近退化。

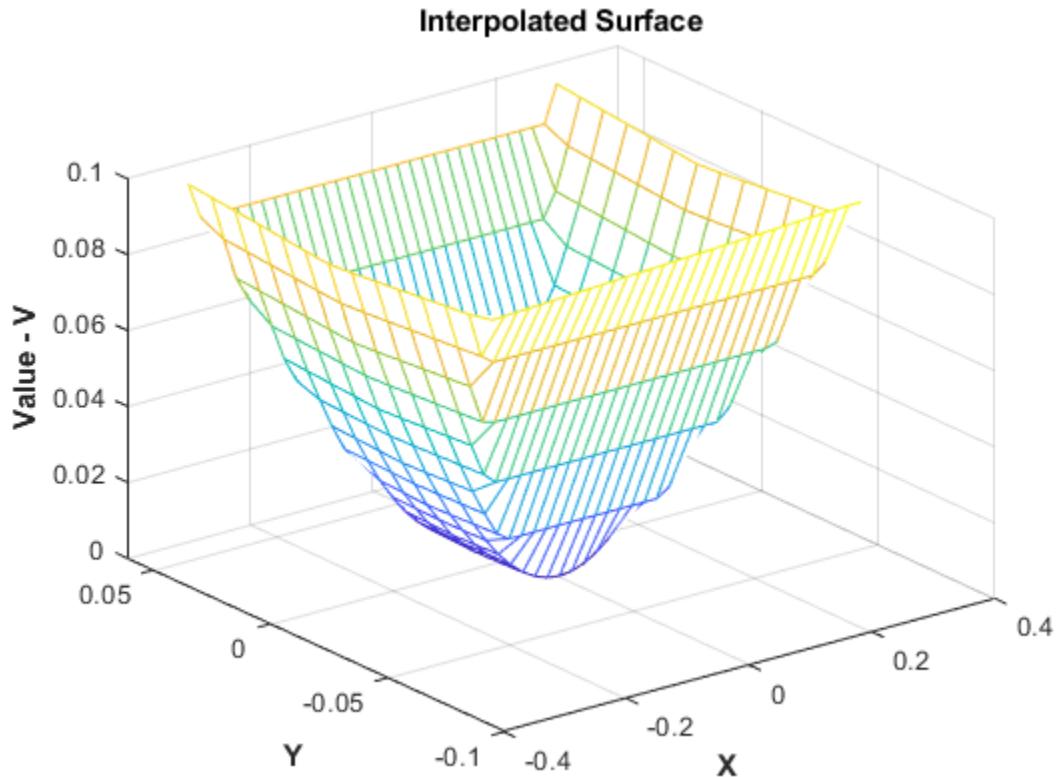
创建一个展示边界附近问题的示例数据集。

```
t = 0.4*pi:0.02:0.6*pi;
x1 = cos(t)';
y1 = sin(t)'-1.02;
x2 = x1;
y2 = y1*(-1);
x3 = linspace(-0.3,0.3,16)';
y3 = zeros(16,1);
x = [x1;x2;x3];
y = [y1;y2;y3];
```

现在将这些样本点提升到曲面 $z = x^2 + y^2$ 上，然后对曲面进行插值。

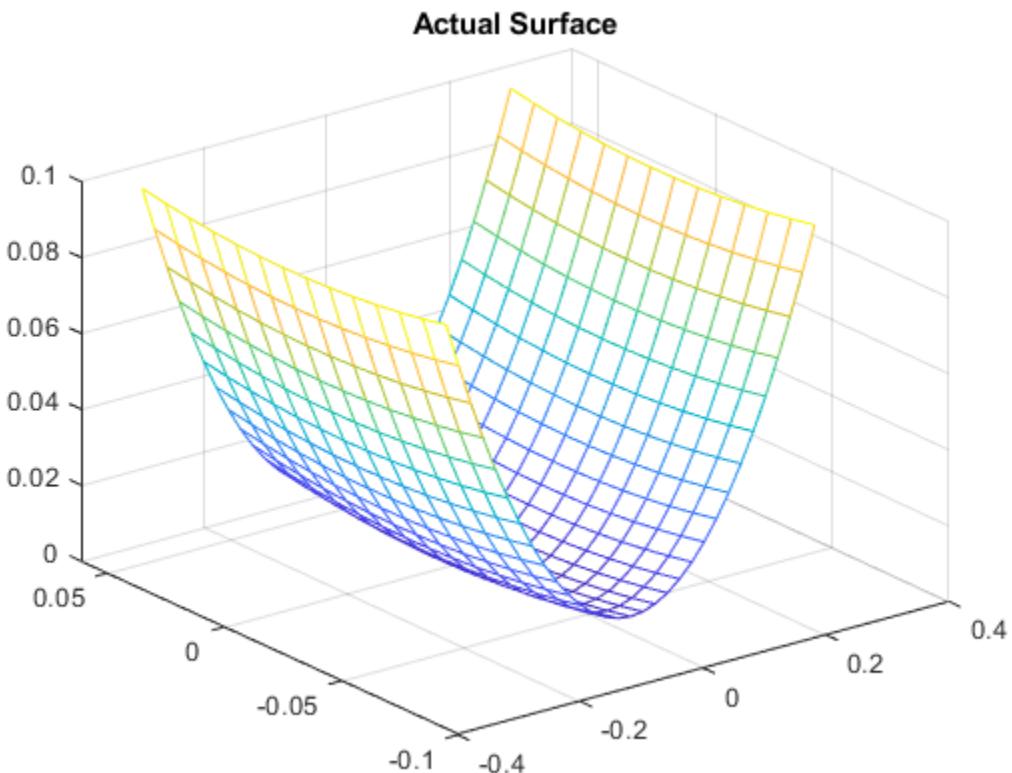
```
z = x.^2 + y.^2;
F = scatteredInterpolant(x,y,z);
[xi,yi] = meshgrid(-0.3:.02:0.3, -0.0688:0.01:0.0688);
```

```
zi = F(xi,yi);
mesh(xi,yi,zi)
xlabel('X','fontweight','b'), ylabel('Y','fontweight','b')
zlabel('Value - V','fontweight','b')
title('Interpolated Surface');
```



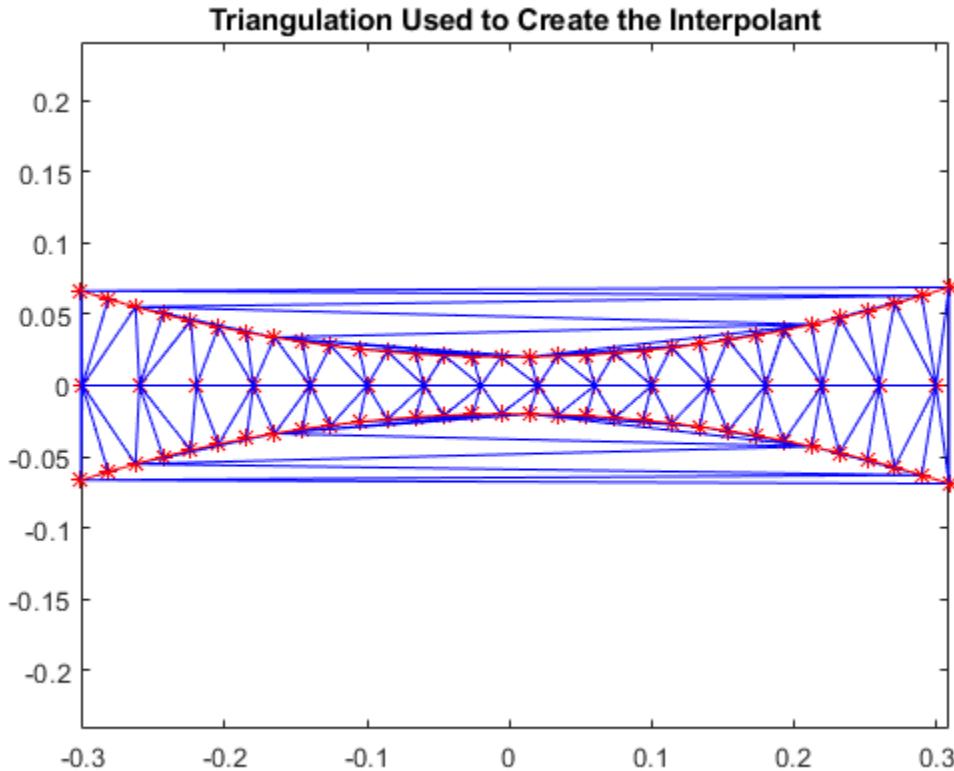
实际曲面是：

```
zi = xi.^2 + yi.^2;
figure
mesh(xi,yi,zi)
title('Actual Surface')
```



要了解插值曲面在边界附近退化的原因，着眼于基本三角剖分会有帮助作用：

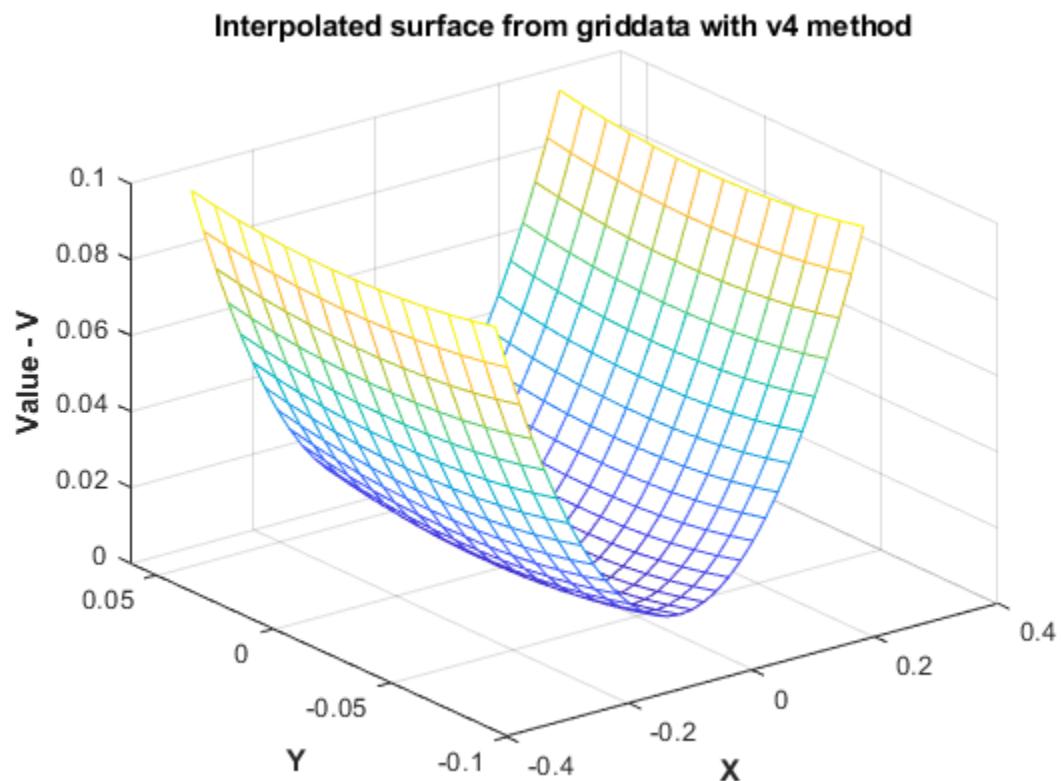
```
dt = delaunayTriangulation(x,y);
figure
plot(x,y,'*r')
axis equal
hold on
triplot(dt)
plot(x1,y1,'-r')
plot(x2,y2,'-r')
title('Triangulation Used to Create the Interpolant')
hold off
```



红色边界内的三角形形状相对良好，它们是根据附近邻域中的点构建的，此区域中的插值情况良好。在红色边界以外，三角形变成类似长条形，连接互相相隔很远的点。没有可精确捕获曲面的充分采样，因此这些区域中的结果不良不足为怪。在三维空间中，目测三角剖分变得更为复杂，但是了解点的分布情况常常有助于说明潜在问题。

MATLAB® 4 `griddata` 方法 '`v4`' 不是基于三角剖分的方法，也不会受到插值曲面在边界附近退化的影响。

```
[xi,yi] = meshgrid(-0.3:.02:0.3, -0.0688:0.01:0.0688);
zi = griddata(x,y,z,xi,yi,'v4');
mesh(xi,yi,zi)
xlabel('X','fontweight','b'), ylabel('Y','fontweight','b')
zlabel('Value - V','fontweight','b')
title('Interpolated surface from griddata with v4 method','fontweight','b');
```



用 'v4' 方法调用 `griddata` 得到的插值曲面与预期的实际曲面相符。

使用特定 Delaunay 三角剖分的插值

本节内容

- “使用 delaunayTriangulation 查询的最近邻点插值”（第 8-59 页）
- “使用 delaunayTriangulation 查询的线性插值”（第 8-59 页）

使用 delaunayTriangulation 查询的最近邻点插值

此示例说明如何使用特定 Delaunay 三角剖分对一组散点执行最近邻点插值。

创建一组二维散点的 delaunayTriangulation。

```
P = -2.5 + 5*gallery('uniformdata',[50 2],0);
DT = delaunayTriangulation(P)
```

```
DT =
delaunayTriangulation with properties:
```

```
    Points: [50x2 double]
    ConnectivityList: [87x3 double]
    Constraints: []
```

在 P 中指定的点处对抛物线函数 V(x,y) 采样。

```
V = P(:,1).^2 + P(:,2).^2;
```

定义 10 个随机查询点。

```
Pq = -2 + 4*gallery('uniformdata',[10 2],1);
```

使用三角剖分 DT 对 V 执行最近邻点插值。使用 nearestNeighbor 计算一组查询点 Pq 的最近邻点顶点 vi 的索引。然后，检查 V 的索引处的特定值。

```
vi = nearestNeighbor(DT,Pq);
Vq = V[vi]
```

Vq = 10×1

```
5.3163
0.3453
4.3026
1.2579
1.9435
5.9194
3.9030
0.4172
11.7282
0.8641
```

使用 delaunayTriangulation 查询的线性插值

此示例说明如何使用特定 Delaunay 三角剖分对一组散点执行线性插值。

可以使用 **triangulation** 的 **pointLocation** 方法计算查询点的封闭三角形和顶点权重的模。该权重称为重心坐标，表示单位体的一部分。即，三个权重的和等于 1。函数 V 在查询点处的插值是 V 在三个顶点处的加权值之和。即，如果函数在三个顶点处具有值 V_1 、 V_2 、 V_3 ，且权重为 B_1 、 B_2 、 B_3 ，则插值为 $(V_1)(B_1) + (V_2)(B_2) + (V_3)(B_3)$ 。

创建一组二维散点的 **delaunayTriangulation**。

```
P = -2.5 + 5*gallery('uniformdata',[50 2],0);
DT = delaunayTriangulation(P)
```

```
DT =
delaunayTriangulation with properties:
```

```
    Points: [50x2 double]
    ConnectivityList: [87x3 double]
    Constraints: []
```

在 P 中的多个点处对抛物线函数 $V(x,y)$ 采样。

```
V = P(:,1).^2 + P(:,2).^2;
```

定义 10 个随机查询点。

```
Pq = -2 + 4*gallery('uniformdata',[10 2],1);
```

使用 **pointLocation** 方法计算封闭每个查询点的三角形。在下面的代码中， ti 包含封闭三角形的 ID， bc 包含与每个三角形关联的重心坐标。

```
[ti,bc] = pointLocation(DT,Pq);
```

计算 $V(x,y)$ 在每个封闭三角形的顶点处的值。

```
triVals = V(DT(ti,:));
```

使用点积计算 $V(x,y)$ 的加权值之和。

```
Vq = dot(bc',triVals)'
```

```
Vq = 10×1
```

```
5.9456
1.1222
4.7963
0.9373
2.3533
3.4219
2.3104
0.7728
8.0479
1.0886
```

另请参阅

[delaunayTriangulation](#) | [nearestNeighbor](#) | [pointLocation](#)

详细信息

- “内插散点数据” (第 8-37 页)

外插散点数据

本节内容

- “影响外插准确性的因素”（第 8-62 页）
- “比较粗略采样和精细采样的散点数据的外插”（第 8-62 页）
- “三维数据外插”（第 8-66 页）

影响外插准确性的因素

`scatteredInterpolant` 提供了对凸包外部的点求近似解值的功能。`'linear'` 外插方法基于凸包边界处梯度的最小平方近似值。为凸包外部的查询点返回的值基于边界处的值和梯度。解的质量取决于数据采样的方式。如果是粗略数据采样，则外插的质量较差。

此外，靠近凸包边界的三角剖分可能具有条形三角形。这些三角形会影响外插结果，就像会影响插值结果一样。有关详细信息，请参阅“凸包附近的插值结果效果不佳”（第 8-54 页）。

应利用您对域外部行为的知识直观检查外插结果。

比较粗略采样和精细采样的散点数据的外插

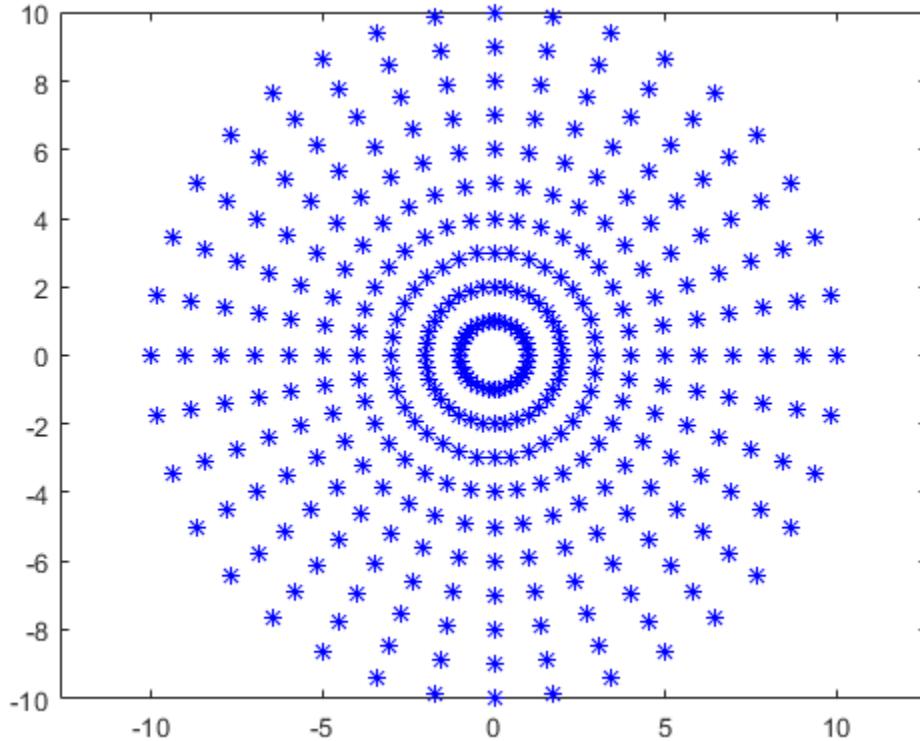
本示例显示如何插入相同抛物线函数的两种不同采样。此示例说明更佳的采样点分布可以生成更好的外插结果。

围绕 10 个同心圆创建间距为 10 度的径向分布点。使用 `bsxfun` 计算坐标 $x = \cos\theta$ 和 $y = \sin\theta$ 。

```
theta = 0:10:350;
c = cosd(theta);
s = sind(theta);
r = 1:10;

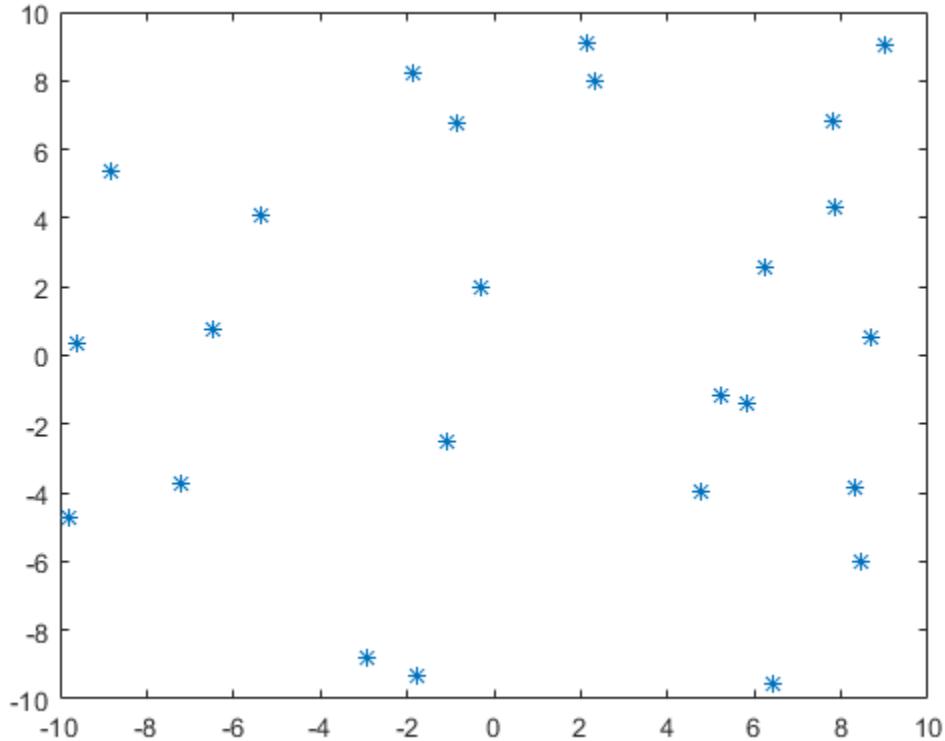
x1 = bsxfun(@times,r.',c);
y1 = bsxfun(@times,r.',s);

figure
plot(x1,y1,'*b')
axis equal
```



创建第二个更粗略分布的点集。使用 gallery 函数在范围 [-10, 10] 中创建随机采样。

```
x2 = -10 + 20*gallery('uniformdata',[25 1],0);
y2 = -10 + 20*gallery('uniformdata',[25 1],1);
figure
plot(x2,y2,'*')
```



在两个点集处对抛物线函数 $v(x,y)$ 采样。

```
v1 = x1.^2 + y1.^2;
v2 = x2.^2 + y2.^2;
```

针对 $v(x,y)$ 的每个采样创建 `scatteredInterpolant`。

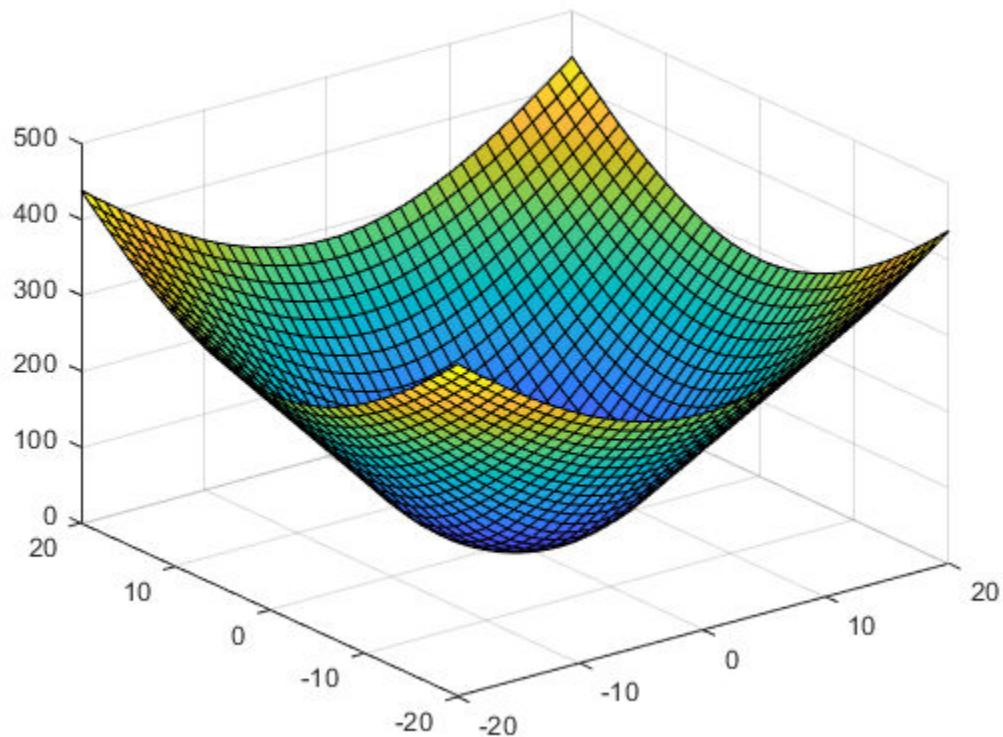
```
F1 = scatteredInterpolant(x1(:,y1(:,v1(:));
F2 = scatteredInterpolant(x2(:,y2(:,v2(:));
```

创建将扩展到每个域外部的查询点网格。

```
[xq,yq] = ndgrid(-20:20);
```

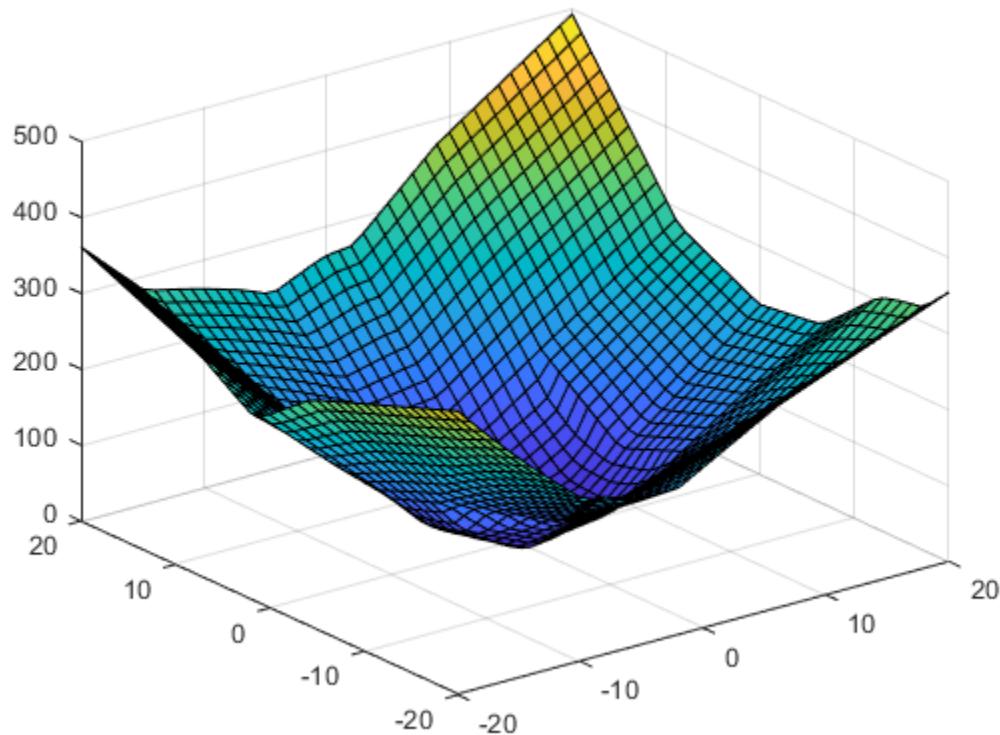
计算 F1 并绘制结果。

```
figure
vq1 = F1(xq,yq);
surf(xq,yq,vq1)
```



计算 F2 并绘制结果。

```
figure  
vq2 = F2(xq,yq);  
surf(xq,yq,vq2)
```



由于对 v_2 中的点进行了粗略采样， F_2 的外插质量不佳。

三维数据外插

此示例说明如何使用 `scatteredInterpolant` 对良好采样的三维网格数据集进行外插。查询点位于完全处于域外部的平面网格上。

创建采样点的 $10 \times 10 \times 10$ 网格。每个维度中的点都处于范围 [-10, 10] 中。

```
[x,y,z] = ndgrid(-10:10);
```

在采样点处对函数 $v(x,y,z)$ 采样。

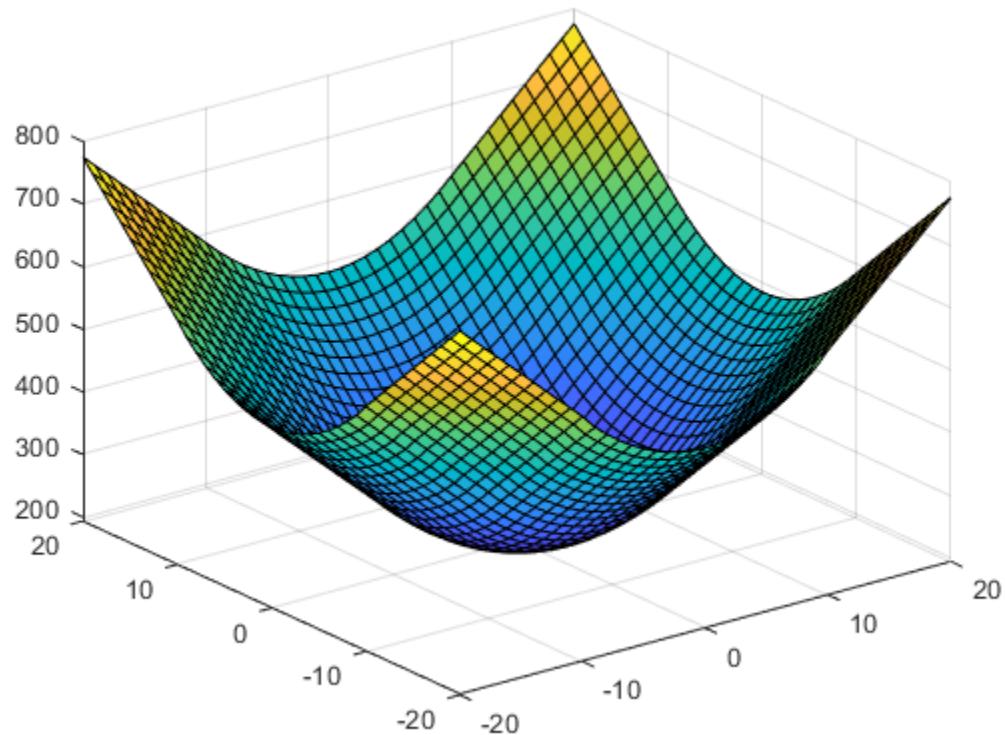
```
v = x.^2 + y.^2 + z.^2;
```

创建 `scatteredInterpolant`，并指定线性插值和外插。

```
F = scatteredInterpolant(x(:,y(:,z(:,v(:,linear',linear');
```

计算 x-y 网格中跨 [-20,20] 范围并且仰角为 $z = 15$ 时的插值。

```
[xq,yq,zq] = ndgrid(-20:20,-20:20,15);
vq = F(xq,yq,zq);
figure
surf(xq,yq,vq)
```



由于对函数进行了良好采样，外插返回较佳结果。

对不同量级的数据进行归一化

此示例说明如何通过归一化来用 `griddata` 改善散点数据插值结果。在某些情况下，归一化可以改善插值结果，但在其他情况下它可能会影响解的精确度。是否使用归一化需要根据插值数据的性质做出判断。

- **优势：**归一化数据在自变量具有不同单位和迥异的规模时可能会改善插值结果。在这种情况下，缩放输入以使其具有类似的量级可以改善插值的数值形态。下面举例说明归一化的好处。假设 x 表示从 500 到 3500 RPM 的引擎速度， y 表示从 0 到 1 的引擎负载。 x 和 y 的规模相差几个数量级，并且它们具有不同单位。
- **注意：**如果各自变量具有相同的单位，则在归一化数据时应小心，即使各变量的规模不同也是如此。对于相同单位的数据，归一化会因添加方向偏差而影响基本三角剖分并最终降低插值的准确度，从而使解失真。例如，如果 x 和 y 都表示位置并且都以“米”为单位，则进行归一化是错误的。此时不建议对 x 和 y 进行不一致的缩放，因为正东 10 米和正北 10 米在空间距离上是相同的。

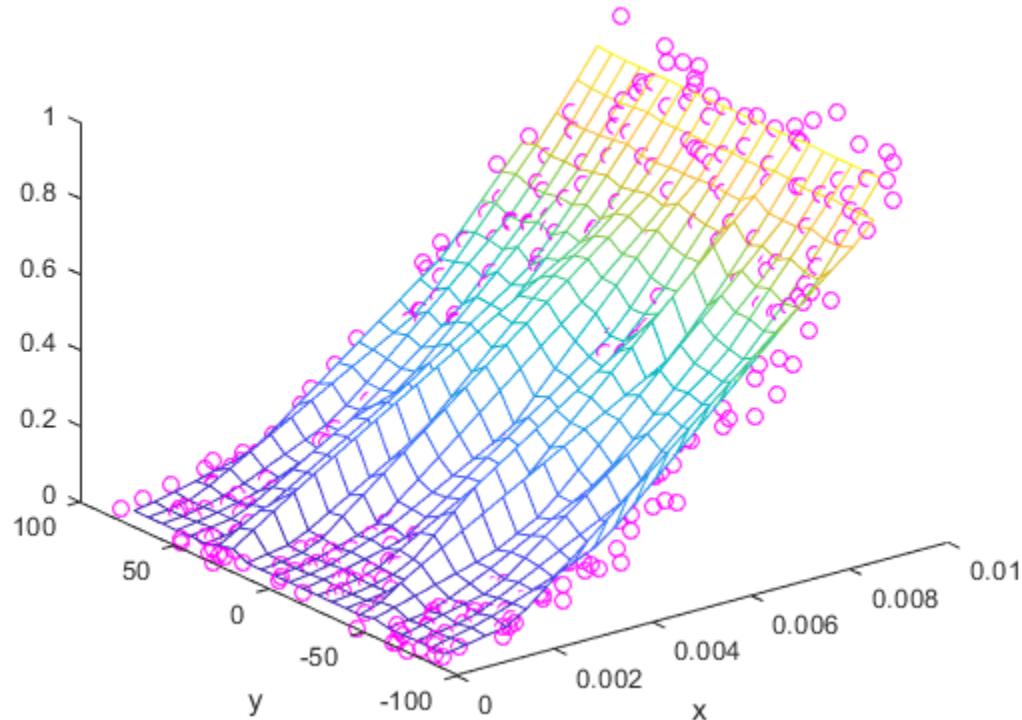
创建一些样本数据，其中 y 中的值比 x 中的值大若干个数量级。假设 x 和 y 具有不同单位。

```
x = rand(1,500)/100;
y = 2.*rand(1,500)-0.5).*90;
z = (x.*1e2).^2;
```

使用以上样本数据构建一个查询点网格。基于网格上的样本数据进行插值并绘制结果。

```
X = linspace(min(x),max(x),25);
Y = linspace(min(y),max(y),25);
[xq, yq] = meshgrid(X,Y);
zq = griddata(x,y,z,xq,yq);

plot3(x,y,z,'mo')
hold on
mesh(xq,yq,zq)
xlabel('x')
ylabel('y')
hold off
```



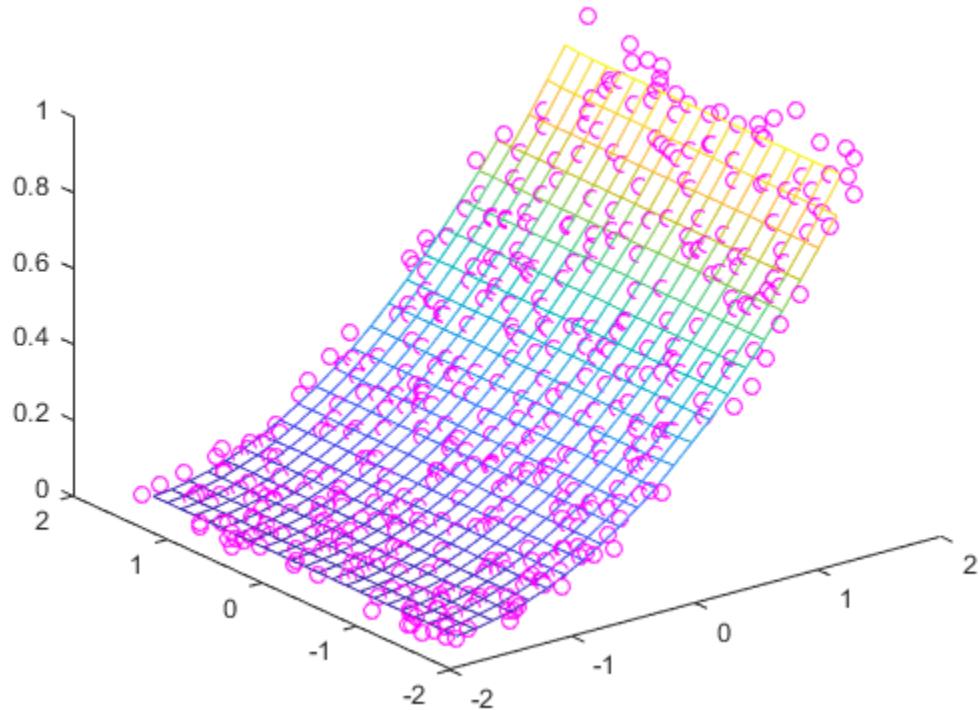
`griddata` 生成的结果不是很平滑，似乎存在含噪数据。这是由于自变量具有不同尺度导致的，因为一个变量的大小细微变化会导致另一个变量的大小发生巨大变化。

由于 `x` 和 `y` 具有不同单位，对它们进行归一化使其具有类似的量级，应该有助于产生更好的结果。使用 `z` 值对样本点进行归一化，并使用 `griddata` 重新生成插值。

```
% Normalize Sample Points
x = normalize(x);
y = normalize(y);

% Regenerate Grid
X = linspace(min(x),max(x),25);
Y = linspace(min(y),max(y),25);
[xq, yq] = meshgrid(X,Y);

% Interpolate and Plot
zq = griddata(x,y,z,xq,yq);
plot3(x,y,z,'mo')
hold on
mesh(xq,yq,zq)
```



在这种情况下，对样本点进行归一化会使 `griddata` 计算出更平滑的解。

另请参阅

`griddata` | `griddatan` | `scatteredInterpolant`

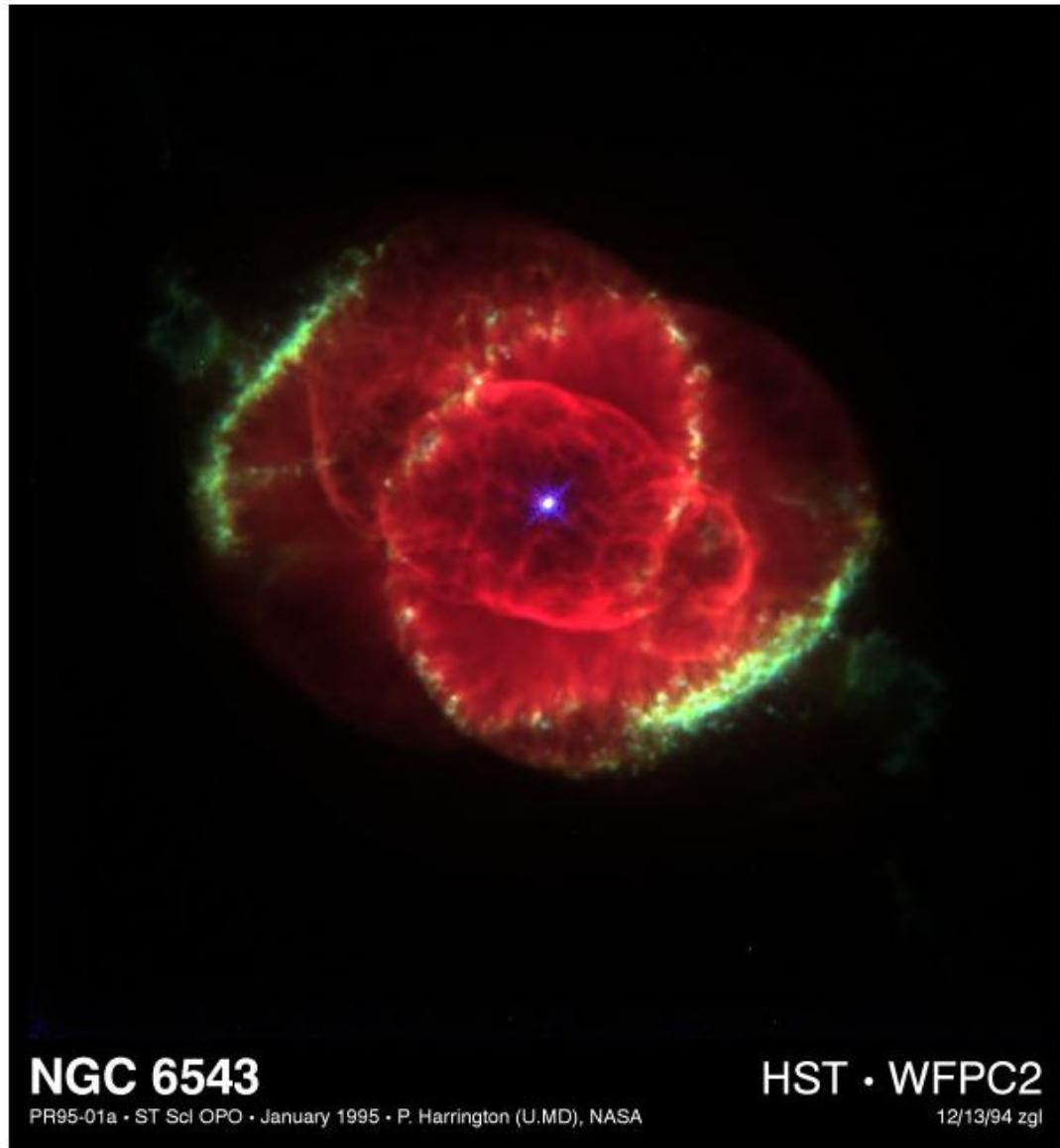
使用网格插值对图像重采样

此示例说明如何使用 `griddedInterpolant` 对图像中的像素重采样。对图像重采样有助于调整分辨率和大小，也可用于在缩放后对像素进行平滑处理。

加载图像

加载并显示图像 `ngc6543a.jpg`，这是哈勃太空望远镜对行星状星云 NGC 6543 拍摄的图像。此图像显示了几个有趣的结构，如同心气体壳、高速气体喷射和异常气体结。表示该图像的矩阵 `A` 是 `uint8` 整数的 $650 \times 600 \times 3$ 矩阵。

```
A = imread('ngc6543a.jpg');
imshow(A)
```



创建插值

为该图像创建一个网格插值对象。对于图像来说，由于像素具有正整数位置，因此适合使用默认网格。鉴于 `griddedInterpolant` 只适用于双精度和单精度矩阵，请将 `uint8` 矩阵转换为双精度。

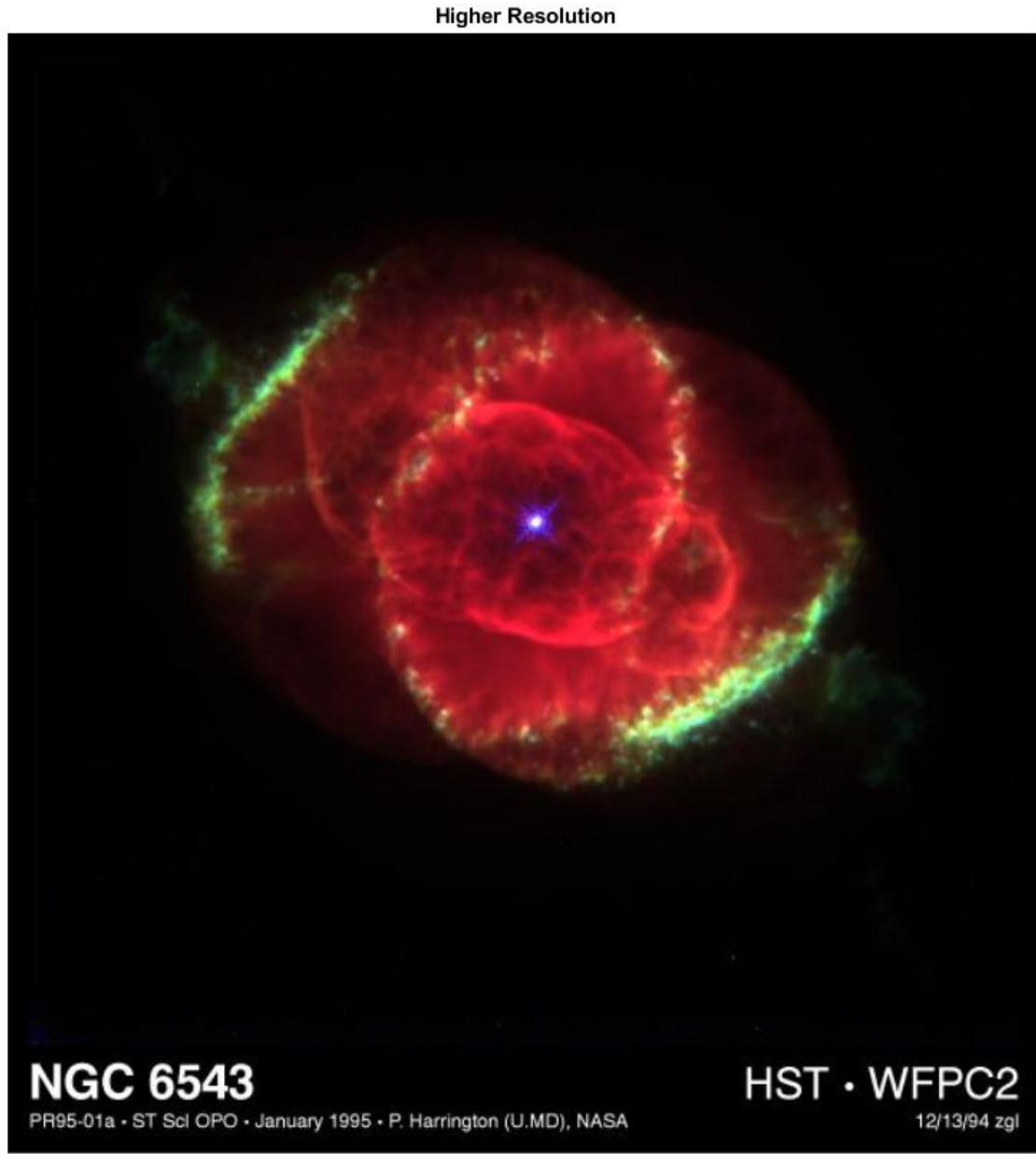
```
F = griddedInterpolant(double(A));
```

对图像像素重采样

当使用大量网格点对图像重采样时，查询插值的最佳方法是使用网格向量。网格向量在元胞数组 $\{xg1, xg2, \dots, xgN\}$ 中组合为列向量。网格向量是表示查询点网格的紧凑方式。使用网格向量，**griddedInterpolant** 不需要形成完整网格即可执行计算。

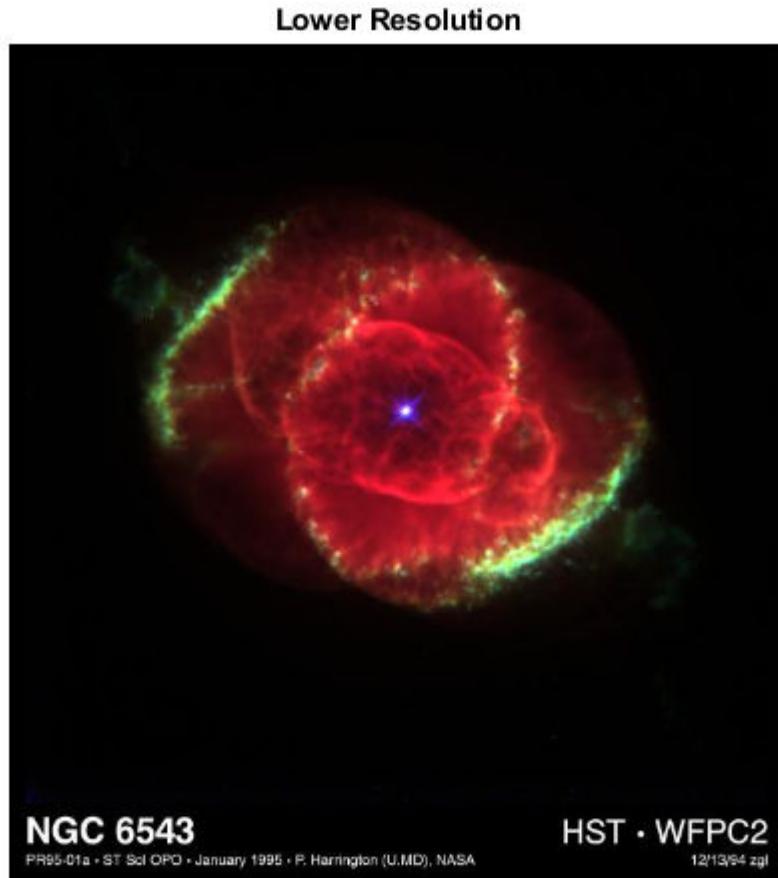
求出原始矩阵维度的大小，并使用这些维度大小对图像重采样，使其大小为原来的 120%。也就是说，对于原始图像中的每 5 个像素，插值图像具有 6 个像素。

```
[sx,sy,sz] = size(A);
xq = (0:5/6:sx)';
yq = (0:5/6:sy)';
zq = (1:sz)';
vq = uint8(F({xq,yq,zq}));
figure
imshow(vq)
title('Higher Resolution')
```



同样，可通过查询比原始图像少 55% 的点的插值来减小图像大小。虽然可以直接对原始图像矩阵进行索引以生成较低分辨率的图像，但是插值使您能够在非整数像素位置对图像重采样。

```
xq = (0:1.55:sx)';
yq = (0:1.55:sy)';
zq = (1:sz)';
vq = uint8(F({xq,yq,zq}));
figure
imshow(vq)
title('Lower Resolution')
```

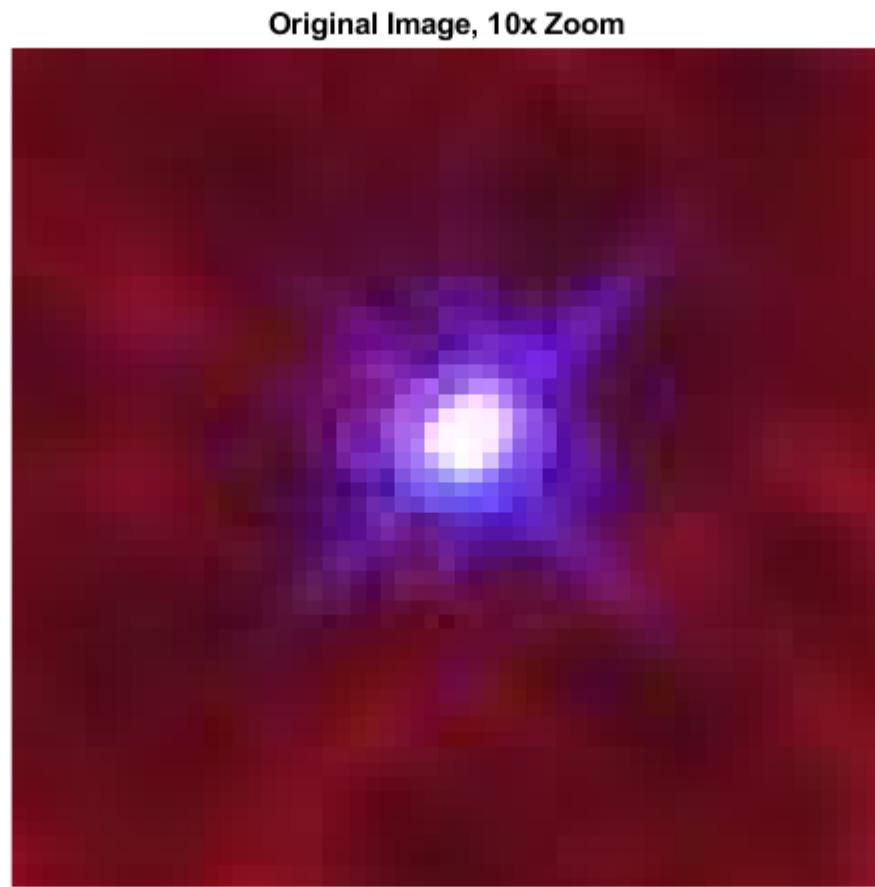


去除缩放伪影

当您放大图像时，目标区域中的像素会变得越来越大，并且图像中的细节很快会丢失。您可以使用图像重采样去除这些缩放伪影。

放大原始图像中心的亮点。（对 A 进行索引是为了使此亮点在图像中居中，以便后续缩放不会将其推出边框外）。

```
imshow(A(1:570,10:600,:),'InitialMagnification','fit')
zoom(10)
title('Original Image, 10x Zoom')
```

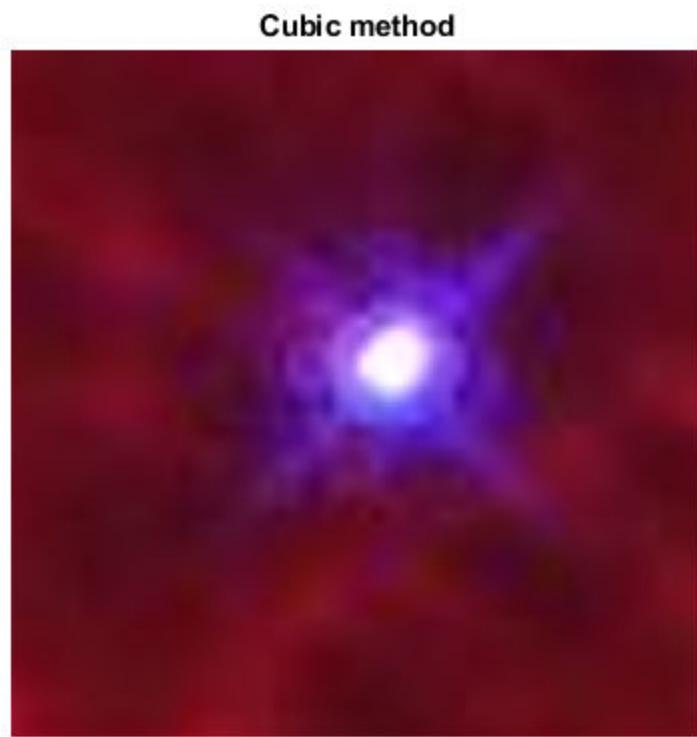


查询插值 F，以大约 10 倍的分辨率重新生成此缩放图像。比较几种不同插值方法的结果。

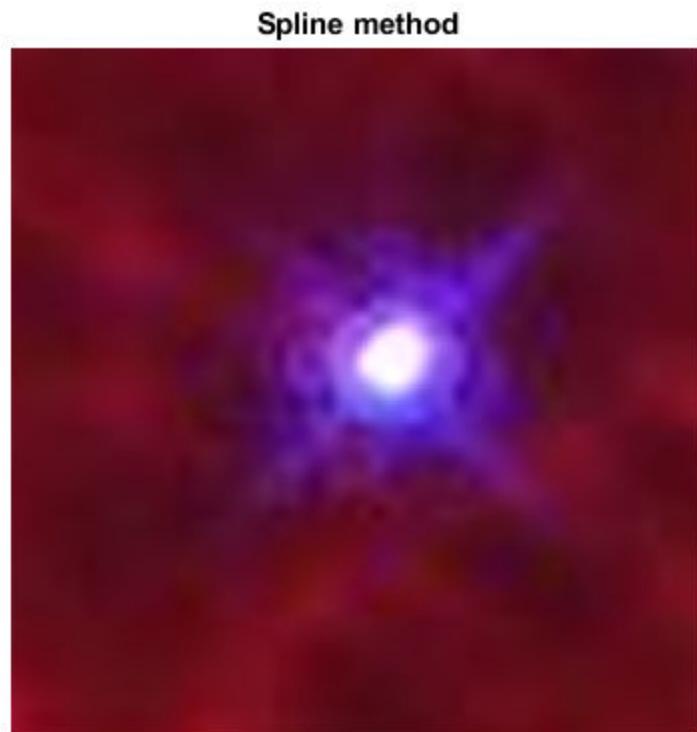
```
xq = (1:0.1:sx)';
yq = (1:0.1:sy)';
zq = (1:sz)';
figure
F.Method = 'linear';
vq = uint8(F({xq,yq,zq}));
imshow(vq(1:5700,150:5900,:),'InitialMagnification','fit')
zoom(10)
title('Linear method')
```



```
figure
F.Method = 'cubic';
vq = uint8(F({xq,yq,zq}));
imshow(vq(1:5700,150:5900,:),'InitialMagnification','fit')
zoom(10)
title('Cubic method')
```



```
figure
F.Method = 'spline';
vq = uint8(F({xq,yq,zq}));
imshow(vq(1:5700,150:5900,:),'InitialMagnification','fit')
zoom(10)
title('Spline method')
```



另请参阅

[griddedInterpolant](#) | [imshow](#)

详细信息

- “插入网格数据” (第 8-3 页)

优化

- “优化非线性函数” (第 9-2 页)
- “通过优化拟合曲线” (第 9-6 页)
- “设置优化选项” (第 9-8 页)
- “优化求解器迭代显示” (第 9-11 页)
- “优化求解器输出函数” (第 9-12 页)
- “优化求解器绘制函数” (第 9-17 页)
- “优化故障排除和提示” (第 9-19 页)

优化非线性函数

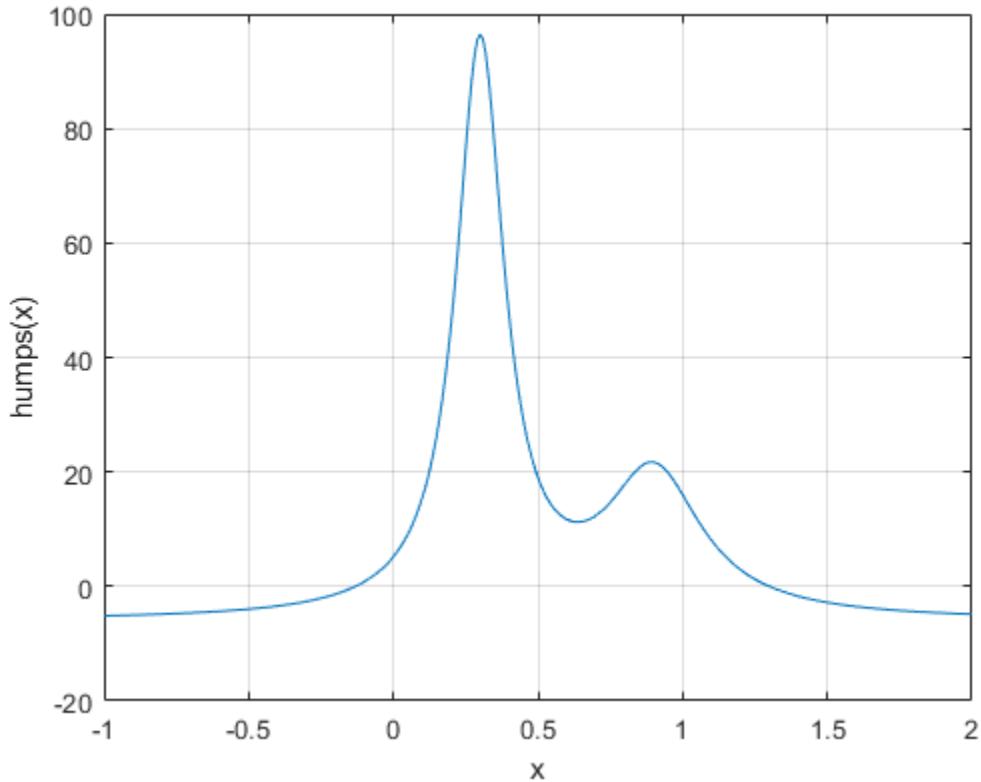
本节内容

- “求一元函数的最小值” (第 9-2 页)
- “求多元函数的最小值” (第 9-3 页)
- “求函数最大值” (第 9-3 页)
- “fminsearch 算法” (第 9-4 页)
- “参考” (第 9-5 页)

求一元函数的最小值

如果给定了一个一元数学函数，可以使用 **fminbnd** 函数求该函数在给定区间中的局部最小值。例如，请考虑 MATLAB® 提供的 **humps.m** 函数。下图显示了 **humps** 的图。

```
x = -1:.01:2;
y = humps(x);
plot(x,y)
xlabel('x')
ylabel('humps(x)')
grid on
```



若要计算 **humps** 函数在 (0.3,1) 范围内的最小值，请使用

```
x = fminbnd(@humps,0.3,1)
```

```
x = 0.6370
```

您可以通过使用 `optimset` 创建选项并将 `Display` 选项设置为 '`iter`' 来查看求解过程的详细信息。将所得选项传递给 `fminbnd`。

```
options = optimset('Display','iter');
x = fminbnd(@humps,0.3,1,options)
```

Func-count	x	f(x)	Procedure
1	0.567376	12.9098	initial
2	0.732624	13.7746	golden
3	0.465248	25.1714	golden
4	0.644416	11.2693	parabolic
5	0.6413	11.2583	parabolic
6	0.637618	11.2529	parabolic
7	0.636985	11.2528	parabolic
8	0.637019	11.2528	parabolic
9	0.637052	11.2528	parabolic

Optimization terminated:

the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-04

```
x = 0.6370
```

这种迭代显示了 `x` 的当前值以及每次计算函数时 `f(x)` 处的函数值。对于 `fminbnd`，一次函数计算对应一次算法迭代。最后一列显示 `fminbnd` 在每次迭代中使用的过程，即黄金分割搜索或抛物线插值。有关详细信息，请参阅“优化求解器迭代显示”（第 9-11 页）。

求多元函数的最小值

`fminsearch` 函数与 `fminbnd` 类似，不同之处在于前者处理多变量函数。请指定起始向量 x_0 ，而非起始区间。`fminsearch` 尝试返回一个向量 x ，该向量是数学函数在此起始向量附近的局部最小值。

要尝试执行 `fminsearch`，请创建一个三元（即 `x`、`y` 和 `z`）函数 `three_var`。

```
function b = three_var(v)
x = v(1);
y = v(2);
z = v(3);
b = x.^2 + 2.5*sin(y) - z^2*x^2*y^2;
```

现在，使用 `x = -0.6`、`y = -1.2` 和 `z = 0.135` 作为起始值求此函数的最小值。

```
v = [-0.6,-1.2,0.135];
a = fminsearch(@three_var,v)
```

```
a =
0.0000 -1.5708 0.1803
```

求函数最大值

`fminbnd` 和 `fminsearch` 求解器尝试求目标函数的最小值。如果具有最大值问题，即，

$$\max_x f(x),$$

然后定义 $g(x) = -f(x)$, 并对 g 取最小值。

例如, 要计算 $\tan(\cos(x))$ 在 $x = 5$ 附近的最大值, 请计算:

```
[x fval] = fminbnd(@(x)-tan(cos(x)),3,8)
```

```
x =
6.2832
```

```
fval =
-1.5574
```

最大值为 1.5574 (报告的 $fval$ 的负值), 并出现在 $x = 6.2832$ 。此答案是正确的, 因为最大值为 $\tan(1) = 1.5574$ (最多五位数), 该值出现在 $x = 2\pi = 6.2832$ 位置。

fminsearch 算法

fminsearch 使用 Lagarias 等人的著作 [1] 中所述的 Nelder-Mead 单纯形算法。此算法对 n 维向量 x 使用 $n + 1$ 个点组成的单纯形。此算法首先向 x_0 添加各分量 $x_0(i)$ 的 5%, 以围绕初始估计值 x_0 生成一个单纯形。然后, 该算法使用上述 n 个向量作为单纯形的除 x_0 之外的元素。(如果 $x_0(i) = 0$, 则算法使用 0.00025 作为分量 i)。然后, 此算法按照以下过程反复修改单纯形。

注意 **fminsearch** 迭代显示方式中的关键字在相应的步骤说明后以**粗体**形式显示。

- 1 用 $x(i)$ 表示当前单纯形中的点列表 $i = 1, \dots, n+1$ 。
- 2 按最小函数值 $f(x(1))$ 到最大函数值 $f(x(n+1))$ 的顺序对单纯形中的点进行排序。在迭代的每个步骤中, 此算法都会放弃当前的最差点 $x(n+1)$ 并接受单纯形中的另一个点。[或者在下面的步骤 7 中, 此算法会更改值大于 $f(x(1))$ 的所有 n 个点。]
- 3 生成反射点

$$r = 2m - x(n+1),$$

其中

$$m = \sum x(i)/n, i = 1 \dots n,$$

并计算 $f(r)$ 。

- 4 如果 $f(x(1)) \leq f(r) < f(x(n))$, 则接受 r 并终止此迭代。**反射**
- 5 如果 $f(r) < f(x(1))$, 则计算延伸点 s

$$s = m + 2(m - x(n+1)),$$

并计算 $f(s)$ 。

a 如果 $f(s) < f(r)$, 接受 s 并终止迭代。**扩展**

b 否则, 接受 r 并终止迭代。**反射**

- 6 如果 $f(r) \geq f(x(n))$, 请在 m 和 $x(n+1)$ 与 r 的较优者之间执行收缩:
- a 如果 $f(r) < f(x(n+1))$ (即 r 优于 $x(n+1)$), 则计算

$$c = m + (r - m)/2$$

并计算 $f(c)$ 。如果 $f(c) < f(r)$, 则接受 c 并终止迭代。**外部收缩**。否则, 继续执行步骤 7 (收缩)。

- b** 如果 $f(r) \geq f(x(n+1))$, 则计算

$$cc = m + (x(n+1) - m)/2$$

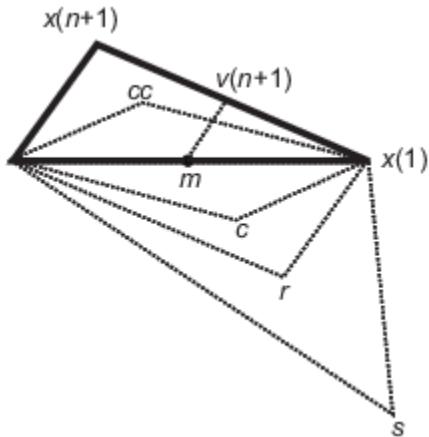
并计算 $f(cc)$ 。如果 $f(cc) < f(x(n+1))$, 则接受 cc 并终止迭代。**内部收缩**。否则, 继续执行步骤 7 (收缩)。

7 计算 n 点

$$v(i) = x(1) + (x(i) - x(1))/2$$

并计算 $f(v(i))$, $i = 2, \dots, n+1$ 。下一迭代中的单纯形为 $x(1), v(2), \dots, v(n+1)$ 。**收缩**

下图显示了 `fminsearch` 可在此过程中计算的点以及每种可能的新单纯形。原始单纯形采用粗体边框。迭代将在符合停止条件之前继续运行。



参考

- [1] Lagarias, J. C., J. A. Reeds, M. H. Wright, and P. E. Wright. "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions." *SIAM Journal of Optimization*, Vol. 9, Number 1, 1998, pp. 112–147.

另请参阅

详细信息

- “优化故障排除和提示” (第 9-19 页)
- “Nonlinear Optimization” (Optimization Toolbox)
- “通过优化拟合曲线” (第 9-6 页)

通过优化拟合曲线

此示例说明如何使用非线性函数对数据进行拟合。在本示例中，非线性函数是标准指数衰减曲线

$$y(t) = A \exp(-\lambda t),$$

其中， $y(t)$ 是时间 t 时的响应， A 和 λ 是要拟合的参数。对曲线进行拟合是指找出能够使误差平方和最小化的参数 A 和 λ

$$\sum_{i=1}^n (y_i - A \exp(-\lambda t_i))^2,$$

其中，时间为 t_i ，响应为 $y_i, i = 1, \dots, n$ 。误差平方和为目标函数。

创建样本数据

通常，您要通过测量获得数据。在此示例中，请基于 $A = 40$ 和 $\lambda = 0.5$ 且带正态分布伪随机误差的模型创建人工数据。

```
rng default % for reproducibility
tdata = 0:0.1:10;
ydata = 40*exp(-0.5*tdata) + randn(size(tdata));
```

编写目标函数

编写一个函数，该函数可接受参数 A 和 λ 以及数据 $tdata$ 和 $ydata$ ，并返回模型 $y(t)$ 的误差平方和。将要优化的所有变量（ A 和 λ ）置入单个向量变量（ x ）。有关详细信息，请参阅“求多元函数的最小值”（第 9-3 页）。

```
type sseval

function sse = sseval(x,tdata,ydata)
A = x(1);
lambda = x(2);
sse = sum((ydata - A*exp(-lambda*tdata)).^2);
```

将此目标函数保存为 MATLAB® 路径上名为 `sseval.m` 的文件。

`fminsearch` 求解器适用于一个变量 x 的函数。但 `sseval` 函数包含三个变量。额外变量 $tdata$ 和 $ydata$ 不是要优化的变量，而是用于优化的数据。将 `fminsearch` 的目标函数定义为仅含有一个变量 x 的函数：

```
fun = @(x)sseval(x,tdata,ydata);
```

有关包括额外参数（例如 $tdata$ 和 $ydata$ ）的信息，请参阅“参数化函数”（第 10-2 页）。

求最优拟合参数

从随机正参数集 $x0$ 开始，使用 `fminsearch` 求使得目标函数值最小的参数。

```
x0 = rand(2,1);
bestx = fminsearch(fun,x0)

bestx = 2×1
```

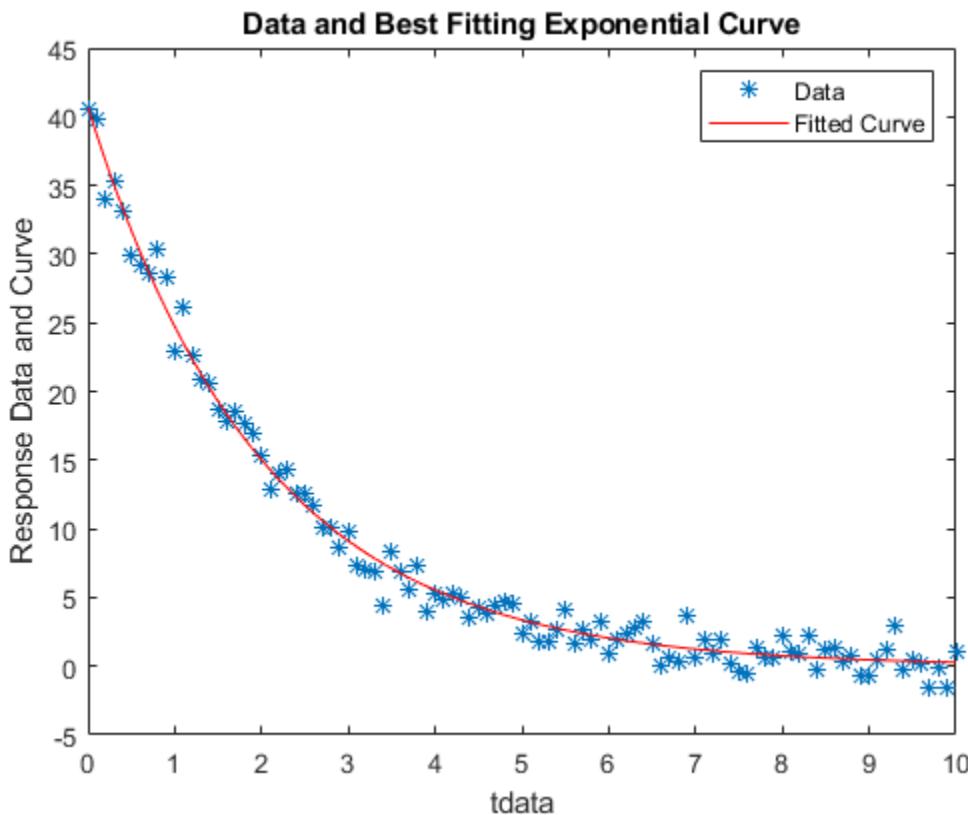
```
40.6877
0.4984
```

结果 `bestx` 与生成数据的参数 $A = 40$ 和 $\lambda = 0.5$ 相当接近。

检查拟合质量

要检查拟合质量，请绘制数据和生成的拟合响应曲线。根据返回的模型参数创建响应曲线。

```
A = bestx(1);
lambda = bestx(2);
yfit = A*exp(-lambda*tdata);
plot(tdata,ydata,'*');
hold on
plot(tdata,yfit,'r');
xlabel('tdata')
ylabel('Response Data and Curve')
title('Data and Best Fitting Exponential Curve')
legend('Data','Fitted Curve')
hold off
```



另请参阅

详细信息

- “优化非线性函数”（第 9-2 页）
- “Nonlinear Regression”（Statistics and Machine Learning Toolbox）

设置优化选项

本节内容
“如何设置选项”（第 9-8 页）
“选项表”（第 9-8 页）
“容差和终止条件”（第 9-9 页）
“输出结构体”（第 9-9 页）

如何设置选项

可以使用由 `optimset` 函数创建的 `options` 结构体来指定优化参数。然后，可以将 `options` 作为输入传递给优化函数，例如，通过使用以下语法调用 `fminbnd`

```
x = fminbnd(fun,x1,x2,options)
```

或使用以下语法调用 `fminsearch`

```
x = fminsearch(fun,x0,options)
```

例如，要显示算法在每次迭代中的输出，请将 `Display` 选项设置为 `'iter'`:

```
options = optimset('Display','iter');
```

选项表

选项	说明	求解器
<code>Display</code>	指示是否在屏幕上显示中间步骤的标签。 <ul style="list-style-type: none"> <code>'notify'</code>（默认值）仅在函数未收敛时显示输出。 <code>'iter'</code> 显示中间步骤（不适用于 <code>lsqnonneg</code>）。请参阅“优化求解器迭代显示”（第 9-11 页）。 <code>'off'</code> 或 <code>'none'</code> 不显示中间步骤。 <code>'final'</code> 仅显示最终输出。 	<code>fminbnd</code> 、 <code>fminsearch</code> 、 <code>fzero</code> 、 <code>lsqnonneg</code>
<code>FunValCheck</code>	检查目标函数值是否有效。 <ul style="list-style-type: none"> 当目标函数或约束返回的值是复数或 <code>Nan</code> 时，<code>'on'</code> 显示错误。 <code>'off'</code>（默认值）不显示错误。 	<code>fminbnd</code> 、 <code>fminsearch</code> 、 <code>fzero</code>
<code>MaxFunEvals</code>	允许的最大函数计算次数。对于 <code>fminbnd</code> ，默认值为 500；对于 <code>fminsearch</code> ，默认值为 <code>200*length(x0)</code> 。	<code>fminbnd</code> 、 <code>fminsearch</code>
<code>MaxIter</code>	允许的最大迭代次数。对于 <code>fminbnd</code> ，默认值为 500；对于 <code>fminsearch</code> ，默认值为 <code>200*length(x0)</code> 。	<code>fminbnd</code> 、 <code>fminsearch</code>
<code>OutputFcn</code>	显示有关求解器的迭代的信息。默认值为 []（无）。请参阅“优化求解器输出函数”（第 9-12 页）。	<code>fminbnd</code> 、 <code>fminsearch</code> 、 <code>fzero</code>
<code>PlotFcns</code>	绘制有关求解器的迭代的信息。默认值为 []（无）。有关可用的预定义函数，请参阅“优化求解器绘制函数”（第 9-17 页）。	<code>fminbnd</code> 、 <code>fminsearch</code> 、 <code>fzero</code>

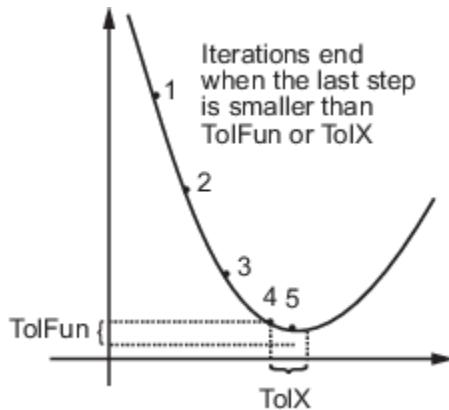
选项	说明	求解器
TolFun	函数值的终止容差。默认值为 1.e-4。请参阅“容差和终止条件”(第 9-9 页)。	fminsearch
TolX	x 的终止容差。默认值为 1.e-4, 但以下两个函数除外: fzero , 其默认值为 eps ($=2.2204e-16$), 以及 lsqnonneg , 其默认值为 $10*eps*norm(c,1)*length(c)$ 。请参阅“容差和终止条件”(第 9-9 页)。	fminbnd、 fminsearch、 fzero、lsqnonneg

容差和终止条件

优化中的迭代次数取决于求解器的终止条件。这些条件包括几个可以设置的容差。一般来说，容差是一个阈值，超过阈值时将终止求解器的迭代。

提示 一般情况下，将 TolFun 和 TolX 容差设置为远高于 eps 并通常高于 1e-14。设置小容差并不能保证得到精确的结果。相反，求解器在收敛时无法识别，并可能继续进行无用的迭代。容差值小于 eps 实际上是禁用了这种终止条件。此技巧不适用于 fzero，它为 TolX 使用默认值 eps。

- TolX 是步长大小的下界，表示 $(x_i - x_{i+1})$ 的范数。如果求解器尝试小于 TolX 的步长，则终止迭代。求解器通常将 TolX 用作相对边界，表示在达到 $|x_i - x_{i+1}| < TolX * (1 + |x_i|)$ 或类似的相对度量时终止迭代。



- TolFun 是步长中目标函数值变化的下边界。如果 $|f(x_i) - f(x_{i+1})| < TolFun$ ，则终止迭代。求解器通常将 TolFun 用作相对边界，表示在达到 $|f(x_i) - f(x_{i+1})| < TolFun(1 + |f(x_i)|)$ 或类似的相对度量时终止迭代。
- MaxIter 是求解器迭代数量的边界。MaxFunEvals 是函数求值数量的边界。

注意 与其他求解器不同，fminsearch 在同时满足 TolFun 和 TolX 时停止运行。

输出结构体

output 结构体包括函数计算次数、迭代次数和算法。当为 fminbnd、fminsearch 或 fzero 提供第四个输出参数时，将显示此结构体，如下所示

```
[x,fval,exitflag,output] = fminbnd(@humps,0.3,1);
```

函数参考页面上提供了每个求解器的 **output** 结构体的详细信息。

output 结构体选项不能选择用于 **optimset**。它是 **fminbnd**、**fminsearch** 和 **fzero** 的可选输出。

另请参阅

详细信息

- “优化非线性函数” (第 9-2 页)
- “优化求解器输出函数” (第 9-12 页)
- “优化求解器绘制函数” (第 9-17 页)

优化求解器迭代显示

通过使用 `optimset` 将 `Display` 选项设置为 '`iter`'，可以获取求解器采取的步骤的详细信息。显示的输出包含以下列表中的标题和项。

标题	显示的信息	求解器
Iteration	迭代次数，表示算法采取的步骤数	<code>fminsearch</code>
Func-count	函数计算的累积数目	<code>fminbnd</code> 、 <code>fminsearch</code> 、 <code>fzero</code>
x	当前点	<code>fminbnd</code> 、 <code>fzero</code>
f(x)	当前目标函数值	<code>fminbnd</code> 、 <code>fzero</code>
min f(x)	找到的最小目标函数值	<code>fminsearch</code>
Procedure	迭代时使用的算法	
	<ul style="list-style-type: none"> • <code>initial</code> • <code>golden</code> (黄金分割搜索) • <code>parabolic</code> (抛物线插值) 	<code>fminbnd</code>
	<ul style="list-style-type: none"> • <code>initial simplex</code> • <code>expand</code> • <code>reflect</code> • <code>shrink</code> • <code>contract inside</code> • <code>contract outside</code> <p>有关详细信息，请参阅 “<code>fminsearch</code> 算法” (第 9-4 页)。</p>	<code>fminsearch</code>
	<ul style="list-style-type: none"> • <code>initial</code> (初始点) • <code>search</code> (搜索包含零的区间) • <code>bisection</code> • <code>interpolation</code> (线性插值或逆二次插值) 	<code>fzero</code>
a, f(a), b, f(b)	搜索具有异号函数值的区间时的搜索点及其函数值	<code>fzero</code>

另请参阅

详细信息

- “设置优化选项” (第 9-8 页)
- “优化求解器输出函数” (第 9-12 页)
- “优化求解器绘制函数” (第 9-17 页)

优化求解器输出函数

本节内容

- “什么是输出函数？”（第 9-12 页）
- “创建和使用输出函数”（第 9-12 页）
- “输出函数的结构体”（第 9-13 页）
- “嵌套输出函数的示例”（第 9-13 页）
- “optimValues 中的字段”（第 9-15 页）
- “算法的状态”（第 9-15 页）
- “Stop 标签”（第 9-16 页）

什么是输出函数？

输出函数是优化函数在算法的每次迭代过程中调用的函数。通常，使用输出函数生成图输出，记录算法生成的数据的历史信息，或者根据当前迭代的数据暂停算法。可以按函数文件、局部函数或嵌套函数的形式创建输出函数。

OutputFcn 选项可以与下列 MATLAB 优化函数配合使用：

- **fminbnd**
- **fminsearch**
- **fzero**

创建和使用输出函数

下面给出了输出函数的一个简单示例，该输出函数绘制优化函数生成的点。

```
function stop = outfun(x, optimValues, state)
stop = false;
hold on;
plot(x(1),x(2),'.');
drawnow
```

在解算以下优化问题时，可以使用此输出函数绘制 **fminsearch** 生成的点

$$\min_x f(x) = \min_x e^{x_1} (4x_1^2 + 2x_2^2 + x_1x_2 + 2x_2).$$

为此，

- 1 创建一个包含前述代码的文件，并将其作为 **outfun.m** 保存在 MATLAB 路径的文件夹中。
- 2 将 **options** 结构体的 **OutputFcn** 字段的值设置为 **outfun** 的函数句柄。

- 3 **options = optimset('OutputFcn', @outfun);**
- 输入以下命令：

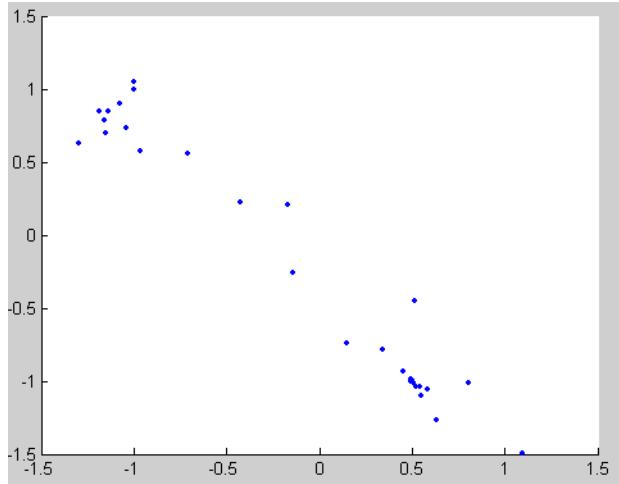
```
hold on
objfun=@(x) exp(x(1))*(4*x(1)^2+2*x(2)^2+x(1)*x(2)+2*x(2));
[x fval] = fminsearch(objfun, [-1 1], options)
hold off
```

这些命令返回解

```
x =
0.1290 -0.5323

fval =
-0.5689
```

并显示以下关于 `fminsearch` 生成的点的绘图：



输出函数的结构体

输出函数的函数定义行采用以下格式：

```
stop = outfun(x, optimValues, state)
```

其中

- `stop` 是一个标志，根据优化例程是停止还是继续，该标志为 `true` 或 `false`。请参阅“Stop 标签”（第 9-16 页）。
- `x` 是算法在当前迭代中计算的点。
- `optimValues` 是包含当前迭代中的数据的结构体。“`optimValues` 中的字段”（第 9-15 页）详细介绍了此结构体。
- `state` 是算法的当前状态。“算法的状态”（第 9-15 页）列出了可能的值。

优化函数在每次迭代中将输入参数的值传递给 `outfun`。

嵌套输出函数的示例

“创建和使用输出函数”（第 9-12 页）中的示例不需要输出函数在每次迭代后保留数据。如果不希望保存各次迭代之间的数据，可以将输出函数编写为函数文件，并从命令行直接调用优化函数。但是，要使输出函数在每次迭代后记录数据，请编写一个用于实现以下目的的文件：

- 以嵌套函数的形式包括输出函数 - 有关详细信息，请参阅 MATLAB 编程基础中的“嵌套函数”。
- 调用优化函数。

在以下示例中，函数文件还包含目标函数作为局部函数。您也可以将目标函数编写为单独的文件或匿名函数。

嵌套函数可以访问其所在的文件中的变量。因此，此方法使输出函数能够在每次迭代后保留变量。

以下示例使用输出函数记录以下求解中的 `fminsearch` 迭代：

$$\min_x f(x) = \min_x e^{x_1} (4x_1^2 + 2x_2^2 + x_1 x_2 + 2x_2).$$

输出函数以矩阵（称为 `history`）的形式返回点序列。

要运行此示例，请执行下列步骤：

- 1 使用 MATLAB 编辑器打开一个新文件。
- 2 将以下代码复制并粘贴到此文件。

```
function [x fval history] = myproblem(x0)
    history = [];
    options = optimset('OutputFcn', @myoutput);
    [x fval] = fminsearch(@objfun, x0,options);

    function stop = myoutput(x,optimvalues,state);
        stop = false;
        if isequal(state,'iter')
            history = [history; x];
        end
    end

    function z = objfun(x)
        z = exp(x(1))*(4*x(1)^2+2*x(2)^2+x(1)*x(2)+2*x(2));
    end
end
```

- 3 将文件作为 `myproblem.m` 保存到 MATLAB 路径上的某个文件夹中。
- 4 在 MATLAB 提示符下，输入

```
[x fval history] = myproblem([-1 1]);
```

函数 `fminsearch` 返回最佳点 `x` 及 `x` 处的目标函数值 `fval`。

```
x,fval
```

```
x =
0.1290 -0.5323

fval =
-0.5689
```

此外，输出函数 `myoutput` 向 MATLAB 工作区返回矩阵 `history`，该矩阵包含算法在每次迭代中生成的点。`history` 的前四行为

```
history(1:4,:)
```

```
ans =
```

```
-1.0000 1.0000
-1.0000 1.0000
```

```
-1.0750  0.9000
-1.0125  0.8500
```

`history` 最后一行的点与最佳点 `x` 相同。

```
history(end,:)
```

```
ans =
```

```
0.1290 -0.5323
```

```
objfun(history(end,:))
```

```
ans =
```

```
-0.5689
```

optimValues 中的字段

下表列出了由优化函数 `fminbnd`、`fminsearch` 和 `fzero` 提供的 `optimValues` 结构体的字段。

表的“命令行显示标题”列中列出了将 `options` 的 `Display` 参数设置为 '`iter`' 时显示的标题。

optimValues 字段 (<code>optimValues.field</code>)	说明	命令行显示标题
<code>funcount</code>	函数计算的累积数目	<code>Func-count</code>
<code>fval</code>	当前点的函数值	<code>min f(x)</code>
<code>iteration</code>	迭代编号 - 从 0 开始	<code>Iteration</code>
<code>procedure</code>	步骤信息	<code>Procedure</code>

算法的状态

下表列出了 `state` 的可能值：

状态	说明
<code>'init'</code>	算法在第一次迭代前处于初始状态。
<code>'interrupt'</code>	算法正在执行迭代。在此状态下，输出函数可停止优化的当前迭代。您可能为了提高计算效率而让输出函数停止迭代。 <code>state</code> 设置为 <code>'interrupt'</code> 后， <code>x</code> 和 <code>optimValues</code> 的值与最后一次调用输出函数时的值相同，在最后一次调用中， <code>state</code> 设置为 <code>'iter'</code> 。
<code>'iter'</code>	算法位于迭代末尾。
<code>'done'</code>	算法在最后一次迭代后处于最终状态。

下面的代码演示输出函数如何使用 `state` 的值来确定要在当前迭代中执行的任务。

```
switch state
  case 'init'
    % Setup for plots or dialog boxes
  case 'iter'
    % Make updates to plots or dialog boxes as needed
  case 'interrupt'
```

```
% Check conditions to see whether optimization  
% should quit  
case 'done'  
    % Cleanup of plots, dialog boxes, or final plot  
end
```

Stop 标签

输出参数 **stop** 是 **true** 或 **false** 的标签。此标志通知优化函数优化是停止 (**true**) 还是继续 (**false**)。下面的示例演示了使用 **stop** 标签的典型方法。

根据 optimValues 中的数据停止优化

输出函数可以根据 **optimValues** 中的当前数据在任何迭代中停止优化。例如，下面的代码在目标函数值小于 5 时将 **stop** 设置为 **true**：

```
function stop = myoutput(x, optimValues, state)  
stop = false;  
% Check if objective function is less than 5.  
if optimValues.fval < 5  
    stop = true;  
end
```

基于对话框输入停止优化

在设计 UI 来执行优化时，可以采用控件（例如**停止**按钮）使输出函数停止优化。以下代码显示如何执行此回调。代码假定**停止**按钮回调将值 **true** 存储在名为 **hObject** 的 **handles** 结构体的 **optimstop** 字段中，而该结构体又存储在 **appdata** 中。

```
function stop = myoutput(x, optimValues, state)  
stop = false;  
% Check if user has requested to stop the optimization.  
stop = getappdata(hObject,'optimstop');
```

另请参阅

详细信息

- “优化求解器绘制函数”（第 9-17 页）
- “设置优化选项”（第 9-8 页）
- “优化求解器迭代显示”（第 9-11 页）

优化求解器绘制函数

本节内容

“什么是绘图函数？”（第 9-17 页）

“示例：绘图函数”（第 9-17 页）

什么是绘图函数？

options 结构体的 **PlotFcns** 字段指定优化函数在每次迭代时调用的一个或多个函数，用于绘制各种进度测度。传递函数句柄或函数句柄的元胞数组。绘图函数的结构体与输出函数的结构体相同。有关此结构体的详细信息，请参阅“优化求解器输出函数”（第 9-12 页）。

PlotFcns 选项可以与下列 MATLAB 优化函数配合使用：

- **fminbnd**
- **fminsearch**
- **fzero**

这些优化函数的预定义绘图函数包括：

- **@optimplotx** 绘制当前点
- **@optimplotfval** 绘制函数值
- **@optimplotfuncount** 绘制函数计数（不适用于 **fzero**）

要查看或修改预定义的绘图函数，请使用 MATLAB 编辑器打开函数文件。例如，要查看函数文件以便绘制当前点，请输入：

```
edit optimplotx.m
```

示例：绘图函数

查看使用 **fminsearch** 与绘图函数 **@optimplotfval** 求最小值的进度：

- 1 为目标函数编写一个文件。在本示例中，使用：

```
function f = onehump(x)
r = x(1)^2 + x(2)^2;
s = exp(-r);
f = x(1)*s+r/20;
```

- 2 设置 **options** 以便使用绘图函数：

```
options = optimset('PlotFcns',@optimplotfval);
```

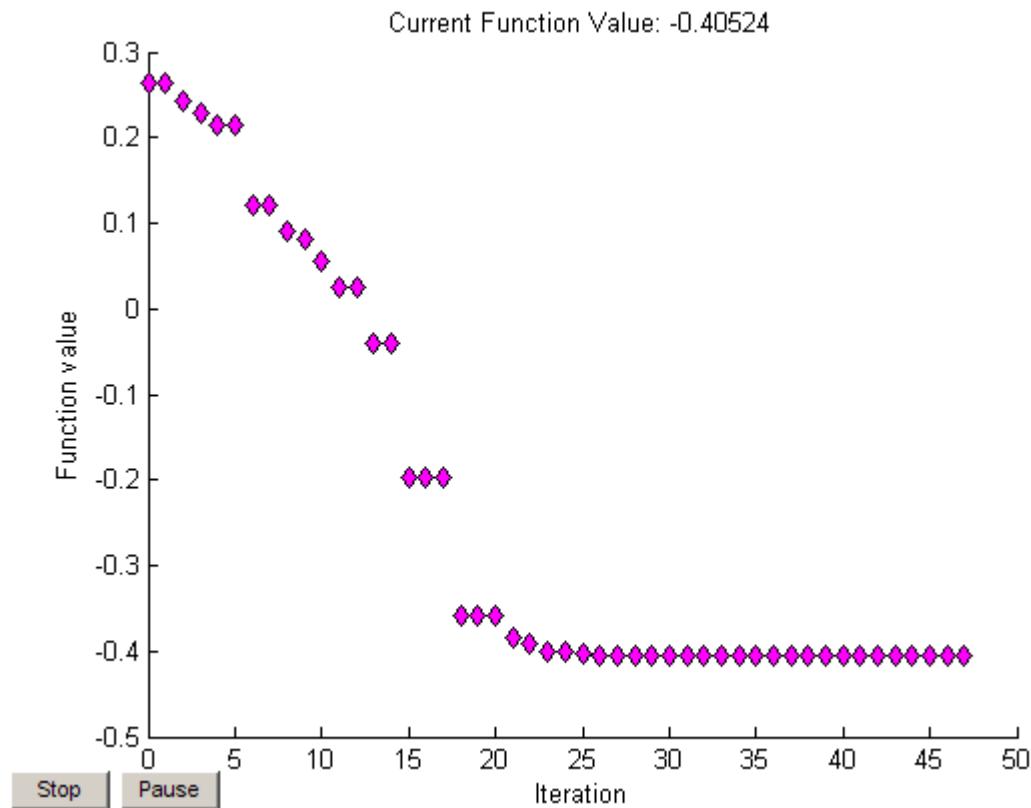
- 3 从 [2,1] 起调用 **fminsearch**：

```
[x ffinal] = fminsearch(@onehump,[2,1],options)
```

- 4 MATLAB 返回以下内容：

```
x =
-0.6691  0.0000
```

```
ffinal =
-0.4052
```



另请参阅

详细信息

- “优化求解器输出函数” (第 9-12 页)
- “设置优化选项” (第 9-8 页)
- “优化求解器迭代显示” (第 9-11 页)

优化故障排除和提示

下表介绍了典型的优化问题并提供了处理建议。

问题	建议
<code>fminbnd</code> 或 <code>fminsearch</code> 求出的解不是全局最小值。全局最小值在搜索空间内的所有点中具有最小的目标函数值。	除非问题是连续的且只有一个最小值，否则无法保证解就是全局最小值。要搜索全局最小值，请从多个起点开始优化（如果使用 <code>fminbnd</code> ，则从多个区间开始优化）。
目标函数 $f(x)$ 在某些点 x 处无法计算出值。这些点称为不可行点。	修改您的函数，以便在不可行点 x 处为 $f(x)$ 返回较大的正值。
最小化例程似乎进入无限循环，或者返回不是最小值的解（对于 <code>fzero</code> ，解不为零）。	可能是您的目标函数返回了 <code>NaN</code> 或复数值。求解器只接受实数目标函数值。任何其他值都可能导致意外结果。要确定是否存在 <code>NaN</code> 或复数值，请设置 <code>options = optimset('FunValCheck','on')</code> 并调用将 <code>options</code> 作为输入参数的优化函数。如果目标函数返回 <code>NaN</code> 或复数值，此设置将导致求解器引发错误。
求解器需要很长时间来求解。	好的起点对大多数优化问题都有帮助。可以根据您的问题特征，在某个区域尝试可能接近解的随机起点。 有时您可以使用演进方法来求解复杂的问题。首先，对具有较少自变量的问题进行求解。然后，通过适当的映射，将较简单问题的解用作更复杂问题的起点。有时候，您还可以在优化问题的初始阶段使用更简单的目标函数和更宽松的停止条件来加速求解过程。
不知道求解器在做什么。	要查看求解器在迭代过程中做了些什么，请执行以下操作： <ul style="list-style-type: none"> 使用绘图函数（如 <code>@optimplotfval</code>）监视迭代。使用 <code>optimset</code> 设置 <code>PlotFcns</code> 选项。请参阅“优化求解器绘制函数”（第 9-17 页）。 设置 <code>Display</code> 选项以便在命令行显示信息。请参阅“优化求解器迭代显示”（第 9-11 页）。
<code>fminsearch</code> 未能求出解。	<code>fminsearch</code> 未能求出解的原因有多种。 <ul style="list-style-type: none"> 缩放不良。如果您的问题没有充分中心化和缩放，求解器可能无法正确收敛。尽量使每个坐标给予目标函数的效应大致相同，并确保接近可能解的每个坐标的比例不要太大或太小。为此，可以编辑目标函数并将每个坐标加上或乘以适当的常量。 终止条件不合适。如果您为 <code>TolFun</code> 或 <code>TolX</code> 指定的值太小，<code>fminsearch</code> 在找到解时可能并未意识到。如果值太大，<code>fminsearch</code> 可能在离解很远的地方就停止了。 初始点不良。尝试从多个初始点开始 <code>fminsearch</code>。 迭代次数不够。如果求解器用尽了所有的迭代次数或卡住，请尝试从终点重新开始 <code>fminsearch</code> 以获得更好的解。有时候，如果您在默认值 <code>200*length(x0)</code> 的基础上增大 <code>MaxFunEvals</code> 和 <code>MaxIter</code> 选项的值，<code>fminsearch</code> 可以得出更好的解。

另请参阅

详细信息

- “优化非线性函数” (第 9-2 页)

函数句柄

参数化函数

本节内容

- “概述” (第 10-2 页)
- “使用嵌套函数参数化” (第 10-2 页)
- “使用匿名函数进行参数化” (第 10-2 页)

概述

本主题向您介绍如何存储或访问向 MATLAB 复合函数 (如 `fzero` 或 `integral`) 传递的数学函数的额外参数。

MATLAB 复合函数基于某个值范围计算数学表达式。这些函数之所以称为复合函数是因为它们是接受函数句柄 (函数的指针) 作为输入的函数。这些函数的每一个都要求目标函数具有特定数量的输入变量。例如, `fzero` 和 `integral` 接受恰好具有一个输入变量的函数的句柄。

假设您需要在系数 `b` 和 `c` 具有不同的值时, 计算三次多项式 $x^3 + bx + c$ 的零点。尽管您可以创建接受三个输入变量 (`x`, `b` 和 `c`) 的函数, 但无法将需要所有这三个输入的函数句柄传递给 `fzero`。不过, 您可以利用匿名函数或嵌套函数的属性来定义其他输入的值。

使用嵌套函数参数化

定义参数的一种方法是使用嵌套函数 - 完全包含于程序文件中另一个函数内的函数。对于此示例, 将创建一个名为 `findzero.m` 的文件, 该文件包含父函数 `findzero` 和嵌套函数 `poly`:

```
function y = findzero(b,c,x0)
y = fzero(@poly,x0);
function y = poly(x)
y = x^3 + b*x + c;
end
end
```

该嵌套函数定义具有一个输入变量 `x` 的三次多项式。父函数接受参数 `b` 和 `c` 作为输入值。将 `poly` 嵌套于 `findzero` 内的原因是, 嵌套函数共享其父函数的工作区。因此, `poly` 函数可以访问您传递给 `findzero` 的 `b` 和 `c` 的值。

要求 `b = 2` 和 `c = 3.5` 时多项式的零点, 如果使用开始点 `x0 = 0`, 则可以从命令行调用 `findzero`:

```
x = findzero(2,3.5,0)
x =
-1.0945
```

使用匿名函数进行参数化

访问额外参数的另一种方法是使用匿名函数。匿名函数是可以在单个命令中定义而无需创建一个单独程序文件的函数。这些函数可以使用当前工作区中可用的任何变量。

例如, 创建用于描述三次多项式的匿名函数的句柄并求零点:

```
b = 2;
c = 3.5;
cubicpoly = @(x) x^3 + b*x + c;
x = fzero(cubicpoly,0)

x =
-1.0945
```

变量 `cubicpoly` 是具有一个输入 `x` 的匿名函数的函数句柄。匿名函数的输入以包含在括号中的形式显示，并紧跟用于创建函数句柄的 `@` 符号之后。由于在您创建 `cubicpoly` 时 `b` 和 `c` 位于工作区中，因此匿名函数不需要这些系数的输入。

无需为匿名函数创建中间变量 `cubicpoly`。可以将函数句柄的整个定义包含在对 `fzero` 的调用中：

```
b = 2;
c = 3.5;
x = fzero(@(x) x^3 + b*x + c,0)

x =
-1.0945
```

您也可以使用匿名函数调用在函数文件中定义的更复杂的目标函数。例如，假设存在具有以下函数定义的名为 `cubicpoly.m` 的文件：

```
function y = cubicpoly(x,b,c)
y = x^3 + b*x + c;
end
```

在命令行上，定义 `b` 和 `c`，然后使用调用 `cubicpoly` 的匿名函数调用 `fzero`：

```
b = 2;
c = 3.5;
x = fzero(@(x) cubicpoly(x,b,c),0)

x =
-1.0945
```

注意 要更改参数的值，必须创建一个新匿名函数。例如：

```
b = 10;
c = 25;
x = fzero(@(x) x^3 + b*x + c,0);
```

另请参阅

详细信息

- “[创建函数句柄](#)”
- “[嵌套函数](#)”
- “[匿名函数](#)”

常微分方程 (ODE)

- “选择 ODE 求解器” (第 11-2 页)
- “ODE 选项摘要” (第 11-9 页)
- “ODE 事件位置” (第 11-11 页)
- “求解非刚性 ODE” (第 11-17 页)
- “解算刚性 ODE” (第 11-21 页)
- “解算微分代数方程 (DAE)” (第 11-27 页)
- “非负 ODE 解” (第 11-32 页)
- “常见 ODE 问题疑难解答” (第 11-35 页)
- “微分方程” (第 11-39 页)
- “求解捕食者-猎物方程” (第 11-48 页)

选择 ODE 求解器

常微分方程

常微分方程 (ODE) 包含与一个自变量 t (通常称为时间) 相关的因变量 y 的一个或多个导数。此处用于表示 y 相对于 t 的导数的表示法对于一阶导数为 y' , 对于二阶导数为 y'' , 依此类推。ODE 的阶数等于 y 在方程中出现的最高阶导数。

例如, 这是一个二阶 ODE:

$$y'' = 9y$$

在初始值问题中, 从初始状态开始解算 ODE。利用初始条件 y_0 以及要在其中求得答案的时间段 (t_0, t_f) , 以迭代方式获取解。在每一步, 求解器都对之前各步的结果应用一个特定算法。在第一个这样的时间步, 初始条件将提供继续积分所需的必要信息。最终结果是, ODE 求解器返回一个时间步向量 $t = [t_0, t_1, t_2, \dots, t_f]$ 以及在每一步对应的解 $y = [y_0, y_1, y_2, \dots, y_f]$ 。

ODE 的类型

MATLAB 中的 ODE 求解器可解算以下类型的一阶 ODE:

- $y' = f(t, y)$ 形式的显式 ODE。
- $M(t, y)y' = f(t, y)$ 形式的线性隐式 ODE, 其中 $M(t, y)$ 为非奇异质量矩阵。该质量矩阵可能为时间或状态相关的矩阵, 也可能为常量矩阵。线性隐式 ODE 涉及在质量矩阵中编码的一阶 y 导数的线性组合。

线性隐式 ODE 可随时变换为显式形式 $y' = M^{-1}(t, y)f(t, y)$ 。不过, 将质量矩阵直接指定给 ODE 求解器可避免这种既不方便还可能带来大量计算开销的变换操作。

- 如果 y' 的某些分量缺失, 则这些方程称为微分代数方程或 DAE, 并且 DAE 方程组会包含一些代数变量。代数变量是导数未出现在方程中的因变量。可通过对方程求导来将 DAE 方程组重写为等效的一阶 ODE 方程组, 以消除代数变量。将 DAE 重写为 ODE 所需的求导次数称为微分指数。**ode15s** 和 **ode23t** 求解器可解算微分指数为 1 的 DAE。
- $f(t, y, y') = 0$ 形式的完全隐式 ODE。完全隐式 ODE 不能重写为显式形式, 还可能包含一些代数变量。**ode15i** 求解器专为完全隐式问题 (包括微分指数为 1 的 DAE) 而设计。

可通过使用 **odeset** 函数创建 **options** 结构体, 来针对某些类型的问题为求解器提供附加信息。

ODE 方程组

您可以指定需要解算的任意数量的 ODE 耦合方程, 原则上, 方程的数量仅受计算机可用内存的限制。如果方程组包含 n 个方程,

$$\begin{pmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_n \end{pmatrix} = \begin{pmatrix} f_1(t, y_1, y_2, \dots, y_n) \\ f_2(t, y_1, y_2, \dots, y_n) \\ \vdots \\ f_n(t, y_1, y_2, \dots, y_n) \end{pmatrix},$$

则用于编写该方程组代码的函数将返回一个向量, 其中包含 n 个元素, 对应于 y'_1, y'_2, \dots, y'_n 值。例如, 考虑以下包含两个方程的方程组

$$\begin{cases} y'_1 = y_2 \\ y'_2 = y_1 y_2 - 2 \end{cases} .$$

用于编写该方程组代码的函数为

```
function dy = myODE(t,y)
dy(1) = y(2);
dy(2) = y(1)*y(2)-2;
```

高阶 ODE

MATLAB ODE 求解器仅可解算一阶方程。您必须使用常规范代换法，将高阶 ODE 重写为等效的一阶方程组

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= y'' \\ &\vdots \\ y_n &= y^{(n-1)}. \end{aligned}$$

这些代换将生成一个包含 n 个一阶方程的方程组

$$\begin{cases} y'_1 = y_2 \\ y'_2 = y_3 \\ \vdots \\ y'_n = f(t, y_1, y_2, \dots, y_n). \end{cases}$$

例如，考虑三阶 ODE

$$y''' - y''y + 1 = 0.$$

使用代换法

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= y'' \end{aligned}$$

生成等效的一阶方程组

$$\begin{cases} y'_1 = y_2 \\ y'_2 = y_3 \\ y'_3 = y_1 y_3 - 1. \end{cases}$$

此方程组的代码则为

```
function dydt = f(t,y)
dydt(1) = y(2);
dydt(2) = y(3);
dydt(3) = y(1)*y(3)-1;
```

复数 ODE

考虑复数 ODE 方程

$$y' = f(t, y),$$

其中 $y = y_1 + iy_2$ 。为解算该方程，需要将实部和虚部分解为不同的解分量，最后重新组合相应的结果。从概念上讲，这类似于

$$\begin{aligned} yv &= [\text{Real}(y) \quad \text{Imag}(y)] \\ fv &= [\text{Real}(f(t, y)) \quad \text{Imag}(f(t, y))] . \end{aligned}$$

例如，如果 ODE 为 $y' = yt + 2i$ ，则可以使用函数文件来表示该方程。

```
function f = complexf(t,y)
% Define function that takes and returns complex values
f = y.*t + 2*i;
```

然后，分解实部和虚部的代码为

```
function fv = imaginaryODE(t,yv)
% Construct y from the real and imaginary components
y = yv(1) + i*yv(2);

% Evaluate the function
yp = complexf(t,y);

% Return real and imaginary in separate components
fv = [real(yp); imag(yp)];
```

在运行求解器以获取解时，初始条件 $y0$ 也会分解为实部和虚部，以提供每个解分量的初始条件。

```
y0 = 1+i;
yv0 = [real(y0); imag(y0)];
tspan = [0 2];
[t,yv] = ode45(@imaginaryODE, tspan, yv0);
```

获得解后，将实部和虚部分量组合到一起可获得最终结果。

```
y = yv(:,1) + i*yv(:,2);
```

基本求解器选择

ode45 适用于大多数 ODE 问题，一般情况下应作为您的首选求解器。但对于精度要求更宽松或更严格的问题而言，**ode23** 和 **ode113** 可能比 **ode45** 更加高效。

一些 ODE 问题具有较高的计算刚度或难度。术语“刚度”无法精确定义，但一般而言，当问题的某个位置存在标度差异时，就会出现刚度。例如，如果 ODE 包含的两个解分量在时间标度上差异极大，则该方程可能是刚性方程。如果非刚性求解器（例如 **ode45**）无法解算某个问题或解算速度极慢，则可以将该问题视为刚性问题。如果您观察到非刚性求解器的速度很慢，请尝试改用 **ode15s** 等刚性求解器。在使用刚性求解器时，可以通过提供 Jacobian 矩阵或其稀疏模式来提高可靠性和效率。

下表提供了关于何时使用每种不同求解器的一般指导原则。

求解器	问题类型	精度	何时使用
ode45	非刚性	中	大多数情况下，您应当首先尝试求解器 ode45 。
ode23		低	对于容差较宽松的问题或在刚度适中的情况下， ode23 可能比 ode45 更加高效。
ode113		低到高	对于具有严格误差容限的问题或在 ODE 函数需要大量计算开销的情况下， ode113 可能比 ode45 更加高效。
ode15s	刚性	低到中	若 ode45 失败或效率低下并且您怀疑面临刚性问题，请尝试 ode15s 。此外，当解算微分代数方程 (DAE) 时，请使用 ode15s 。
ode23s		低	对于误差容限较宽松的问题， ode23s 可能比 ode15s 更加高效。它可以解算一些刚性问题，而使用 ode15s 解算这些问题的效率不高。 ode23s 会在每一步计算 Jacobian，因此通过 odeset 提供 Jacobian 有利于最大限度地提高效率和精度。 如果存在质量矩阵，则它必须为常量矩阵。
ode23t		低	对于仅仅是刚度适中的问题，并且您需要没有数值阻尼的解，请使用 ode23t 。 ode23t 可解算微分代数方程 (DAE)。
ode23tb		低	与 ode23s 一样，对于误差容限较宽松的问题， ode23tb 求解器可能比 ode15s 更加高效。
ode15i	完全隐式	低	对于完全隐式问题 $f(t,y,y') = 0$ 和微分指数为 1 的微分代数方程 (DAE)，请使用 ode15i 。

有关何时使用每种求解器的详细信息和更多建议，请参阅 [5]。

ODE 示例和文件摘要

有几个示例文件可用作大多数 ODE 问题的有用起点。要运行**微分方程示例**应用，以便轻松浏览和运行示例，请键入

odeexamples

要打开单独的示例文件进行编辑，请键入

edit exampleFileName.m

要运行示例，请键入

exampleFileName

此表包含可用的 ODE 和 DAE 示例文件及其使用的求解器和选项的列表。其中包含示例子集的链接，这些示例也已直接发布在文档中。

示例文件	使用的求解器	指定的选项	说明	文档链接
amp1d ae	ode23t	• 'Mass'	刚性 DAE - 包含常量奇异质量矩阵的电路	"解算刚性微分代数方程"
ballode	ode23	• 'Events' • 'OutputFcn' • 'OutputSel' • 'Refine' • 'InitialStep' • 'MaxStep'	简单事件位置 - 弹球	"ODE 事件位置" (第 11-11 页)
batonde	ode45	• 'Mass'	与时间和状态相关的质量矩阵的 ODE - 短棒的移动	-
brussoode	ode15s	• 'JPattern' • 'Vectorized'	刚性大问题 - 化学反应中的扩散 (Brusselator)	"解算刚性 ODE" (第 11-21 页)
burgersode	ode15s	• 'Mass' • 'MStateDependence' • 'JPattern' • 'MvPattern' • 'RelTol' • 'AbsTol'	强烈依赖于状态的质量矩阵的 ODE - 使用移动网格方法解算 Burgers 方程	-
fem1ode	ode15s	• 'Mass' • 'MStateDependence' • 'Jacobian'	与时间相关的质量矩阵的刚性问题 - 有限元方法	-

示例文件	使用的求解器	指定的选项	说明	文档链接
fem2ode	ode23s	• 'Mass'	常量质量矩阵的刚性问题 - 有限元方法	-
hb1ode	ode15s	-	在非常长的区间内解算的刚性 ODE 问题 - Robertson 化学反应	-
hb1dae	ode15s	• 'Mass' • 'RelTol' • 'AbsTol' • 'Vectorized'	基于守恒定律的刚性线性隐式 DAE - Robertson 化学反应	"将 Robertson 问题作为半隐式微分代数方程 (DAE) 求解"
ihb1dae	ode15i	• 'RelTol' • 'AbsTol' • 'Jacobian'	刚性完全隐式 DAE - Robertson 化学反应	"将 Robertson 问题作为隐式微分代数方程 (DAE) 求解"
iburgersode	ode15i	• 'RelTol' • 'AbsTol' • 'Jacobian' • 'JPattern'	隐式 ODE 方程组 - Burgers 方程	-
kneede	ode15s	• 'NonNegative'	具有非负约束的 "膝盖问题"	"非负 ODE 解" (第 11-32 页)
orbitode	ode45	• 'RelTol' • 'AbsTol' • 'Events' • 'OutputFcn'	高级事件位置 - 限制性三体问题	"ODE 事件位置" (第 11-11 页)
rígido de	ode45	-	非刚性问题 - 刚体在不受外力作用时的欧拉方程	"求解非刚性 ODE" (第 11-17 页)
vdpode	ode15s	• 'Jacobian'	可参数化的 van der Pol 方程 (对于较大 μ , 为刚性问题)	"解算刚性 ODE" (第 11-21 页)

参考

- [1] Shampine, L. F. and M. K. Gordon, *Computer Solution of Ordinary Differential Equations: the Initial Value Problem*, W. H. Freeman, San Francisco, 1975.
- [2] Forsythe, G., M. Malcolm, and C. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, New Jersey, 1977.
- [3] Kahaner, D., C. Moler, and S. Nash, *Numerical Methods and Software*, Prentice-Hall, New Jersey, 1989.
- [4] Shampine, L. F., *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, New York, 1994.
- [5] Shampine, L. F. and M. W. Reichelt, "The MATLAB ODE Suite," *SIAM Journal on Scientific Computing*, Vol. 18, 1997, pp. 1-22.

[6] Shampine, L. F., Gladwell, I. and S. Thompson, *Solving ODEs with MATLAB*, Cambridge University Press, Cambridge UK, 2003.

另请参阅

[odeset](#) | [odextend](#)

详细信息

- “求解非刚性 ODE” (第 11-17 页)
- “解算刚性 ODE” (第 11-21 页)
- “解算微分代数方程 (DAE)” (第 11-27 页)

外部网站

- 常微分方程

ODE 选项摘要

解算 ODE 经常要求微调参数、调整误差容限或向求解器传递附加信息。本主题说明如何指定选项以及每个选项与哪些微分方程求解器兼容。

选项语法

使用 `odeset` 函数创建 `options` 结构体，然后将其作为第四个输入参数传递给求解器。例如，要调整相对和绝对误差容限，请使用以下命令：

```
opts = odeset('RelTol',1e-2,'AbsTol',1e-5);
[t,y] = ode45(@odefun,tspan,y0,opts);
```

如果您使用不带输入参数的 `odeset` 命令，MATLAB 将显示每个选项的可能值列表，并以花括号 {} 表示默认值。

`odeget` 函数可以查询现有结构体中的选项值，您可以用该函数来根据条件动态更改选项值。例如，以下代码检测 `Stats` 是否设置为 'on'，并根据需要更改其值：

```
if isempty(odeget(opts,'Stats'))
    odeset(opts,'Stats','on')
end
```

选项与每个求解器的兼容性

在 `odeset` 中，有些选项是通用的，可与任何求解器兼容，还有一些选项是特定于求解器的。下表总结了每个选项与不同求解器之间的兼容性。

选项	ode45	ode23	ode113	ode15s	ode23s	ode23t	ode23tb	ode15i
RelTol	✓	✓	✓	✓	✓	✓	✓	✓
AbsTol								
NormCo ntrol								
OutputF cn	✓	✓	✓	✓	✓	✓	✓	✓
OutputS el								
Refine								
Stats								
NonNeg ative	✓	✓	✓	✓*	-	✓*	✓*	-
Events	✓	✓	✓	✓	✓	✓	✓	✓**
MaxStep	✓	✓	✓	✓	✓	✓	✓	✓
InitialSt ep								

选项	ode45	ode23	ode113	ode15s	ode23s	ode23t	ode23tb	ode15i
Jacobian JPattern Vectorized	-	-	-	✓	✓	✓	✓	✓
Mass MStateDependence MvPattern MassSingular	✓ ✓ — —	✓ ✓ — —	✓ ✓ — —	✓ ✓ ✓ ✓	✓ — — —	✓ ✓ ✓ ✓	✓ ✓ ✓ —	- - - -
InitialSlope	-	-	-	✓	-	✓	-	-
MaxOrder BDF	-	-	-	✓	-	-	-	✓ —

* 只有在解算不涉及质量矩阵的问题时，才对 `ode15s`、`ode23t` 和 `ode23tb` 求解器使用 `NonNegative` 参数。

** `ode15i` 的事件函数必须接受 `yp` 的第三个输入参数。

用法示例

MATLAB 包含几个示例文件，说明了如何使用各种选项。例如，键入 `edit ballode` 查看使用 'Events' 指定事件函数的示例，或键入 `edit batonode` 查看使用 'Mass' 指定质量矩阵的示例。有关示例文件及其使用的选项的完整总结，请参阅 “ODE 示例和文件摘要”（第 11-6 页）。

另请参阅

`odeget` | `odeset`

详细信息

- “选择 ODE 求解器”（第 11-2 页）
- “ODE 事件位置”（第 11-11 页）
- “非负 ODE 解”（第 11-32 页）

ODE 事件位置

本节内容

- “什么是事件位置？”（第 11-11 页）
- “编写事件函数”（第 11-11 页）
- “事件信息”（第 11-12 页）
- “局限性”（第 11-12 页）
- “简单事件位置：弹球”（第 11-12 页）
- “高级事件位置：限制性三体问题”（第 11-13 页）

什么是事件位置？

某些 ODE 方程组的解算难度部分在于确定停止求解的合适时间。积分区间中的最终时间可能由具体事件（而非数字）进行定义。从树上落下的苹果便是一个示例。ODE 求解器应当在苹果落地后立即停止，但您事前并不知道该事件会在何时发生。与之相似地，有些问题所涉及的事件则不会终止求解。沿行星轨道运行的卫星便是一个示例。这种情况下，您可能不希望提前停止积分，但仍希望检测到卫星每次围绕较大天体完成一个运行周期的时刻。

在 ODE 的求解过程中，使用事件函数来检测发生特定事件的时刻。事件函数采用您指定的表达式，并在该表达式等于零时检测事件。它们还能在检测到事件时提示 ODE 求解器停止积分。

编写事件函数

使用 `odeset` 函数的 'Events' 选项指定事件函数。事件函数必须具有一般形式

```
[value,isterminal,direction] = myEventsFcn(t,y)
```

在 `ode15i` 的情况下，事件函数还必须接受 `yp` 的第三个输入参数。

输出参数 `value`、`isterminal` 和 `direction` 均为向量，其第 *i* 个元素与第 *i* 个事件相对应：

- `value(i)` 是描述第 *i* 个事件的数学表达式。当 `value(i)` 等于零时发生事件。
- 如果当第 *i* 个事件发生时停止积分，则 `isterminal(i) = 1`。否则为 `0`。
- 如果需要查找全零值（默认值），则 `direction(i) = 0`。值为 `+1` 时仅在事件函数递增的位置查找零，值为 `-1` 时仅在事件函数递减的位置查找零。指定 `direction = []` 将对所有事件使用默认值 `0`。

再次考虑苹果从树上落下的情形。表示落体的 ODE 为

$$y'' = -1 + y^2,$$

初始条件为 $y(0) = 1$ 和 $y'(0) = 0$ 。您可以使用事件函数来确定 $y(t) = 0$ 的时刻，即苹果落地的时刻。对于此问题，检测苹果何时落地的事件函数为

```
function [position,isterminal,direction] = appleEventsFcn(t,y)
position = y(1); % The value that we want to be zero
isterminal = 1; % Halt integration
direction = 0; % The zero can be approached from either direction
```

事件信息

如果指定了事件函数，则使用三个额外的输出参数调用 ODE 求解器，如下所示

```
[t,y,te,ye,ie] = odeXY(odefun,tspan,y0,options)
```

求解器返回的三个附加输出对应于检测到的事件：

- **te** 是发生事件的时刻的列向量。
- **ye** 包含 **te** 中的每个事件时刻对应的解值。
- **ie** 包含事件函数返回的向量的索引。这些值指示求解器检测到的事件。

您也可以使用单个输出调用求解器，如下所示

```
sol = odeXY(odefun,tspan,y0,options)
```

这种情况下，事件信息以 **sol.te**、**sol.ye** 和 **sol.ie** 的形式存储在结构体中。

局限性

ODE 求解器采用的与事件函数配合使用的求根机制存在下列局限性：

- 如果在积分的第一步即发生终止事件，则求解器会将该事件记录为非终止事件并继续积分。
- 如果在第一步发生多个终止事件，则仅记录第一个事件，并且求解器会继续积分。
- 零值由每步之间的符号交叉确定。因此，对于两步间有偶数个交叉的函数而言，可能会错失其零值。

如果求解器步长跨越了多个事件，请尝试减小 **RelTol** 和 **AbsTol** 以提高精度。也可以设置 **MaxStep** 以便为步长大小设置上限。调整 **tspan** 不会更改求解器所用的步长。

简单事件位置：弹球

此示例说明如何编写一个与 ODE 求解器配合使用的简单事件函数。示例文件 **ballode** 将模拟弹球的运动。事件函数在球每次弹起时停止积分，然后使用新的初始条件重新开始积分。在球的弹跳过程中，积分多次停止并重新开始。

弹球的方程为

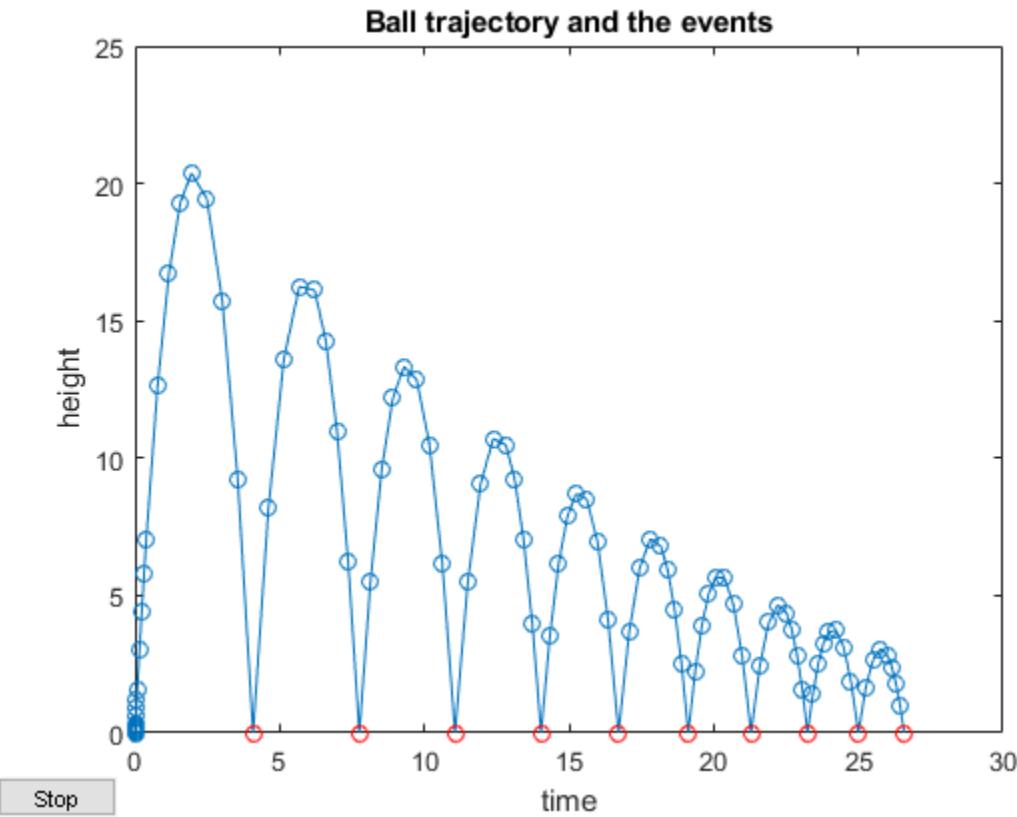
$$\begin{aligned}y'_1 &= y_2 \\y'_2 &= -9.8.\end{aligned}$$

当球的高度 $y_1(t)$ 在逐渐减小后等于零时，发生球弹起。为此行为编码的事件函数为

```
function [value,isterminal,direction] = bounceEvents(t,y)
value = y(1); % Detect height = 0
isterminal = 1; % Stop the integration
direction = -1; % Negative direction only
```

键入 **ballode** 以运行该示例并演示使用事件函数模拟球弹跳的过程。

```
ballode
```



高级事件位置：限制性三体问题

此示例说明如何使用事件函数的定向分量。示例文件 `orbitode` 模拟限制性三体问题，其中一个主体环绕两个大得多的主体做轨道运行。事件函数将确定轨道中距离环绕主体最近和最远的点。由于事件函数在轨道最近点和最远点的值相同，因此将使用过零的方向来区分二者。

限制性三体问题的方程为

$$\begin{aligned} y'_1 &= y_3 \\ y'_2 &= y_4 \\ y'_3 &= 2y_4 + y_1 - \frac{\mu^*(y_1 + \mu)}{r_1^3} - \frac{\mu(y_1 - \mu^*)}{r_2^3} \\ y'_4 &= -2y_3 + y_2 - \frac{\mu^*y_3}{r_1^3} - \frac{\mu y_3}{r_2^3}, \end{aligned}$$

其中

$$\begin{aligned} \mu &= 1/82.45 \\ \mu^* &= 1 - \mu \\ r_1 &= \sqrt{(y_1 + \mu)^2 + y_2^2} \\ r_2 &= \sqrt{(y_1 - \mu^*)^2 + y_2^2}. \end{aligned}$$

前两个解分量是微小物体的坐标，因此针对一个分量绘制另一个分量可以得到物体的轨迹。

`orbitode.m` 中嵌套的事件函数将搜索两个事件。一个事件查找距离起点最近的点，另一个事件查找宇宙飞船返回到起点的点。即使积分器使用的步长并非通过事件位置确定，也会准确定位事件。在此示例中，

指定过零方向的功能非常重要。返回到起点的点和距离起点最远的点具有相同的事件值，并由交叉方向来区分这两个点。为此行为编码的事件函数为

```
function [value,isterminal,direction] = orbitEvents(t,y)
% dDSQdt is the derivative of the equation for current distance. Local
% minimum/maximum occurs when this value is zero.
dDSQdt = 2 * ((y(1:2)-y0(1:2))' * y(3:4));
value = [dDSQdt; dDSQdt];
isterminal = [1; 0];      % stop at local minimum
direction = [1; -1];      % [local minimum, local maximum]
end
```

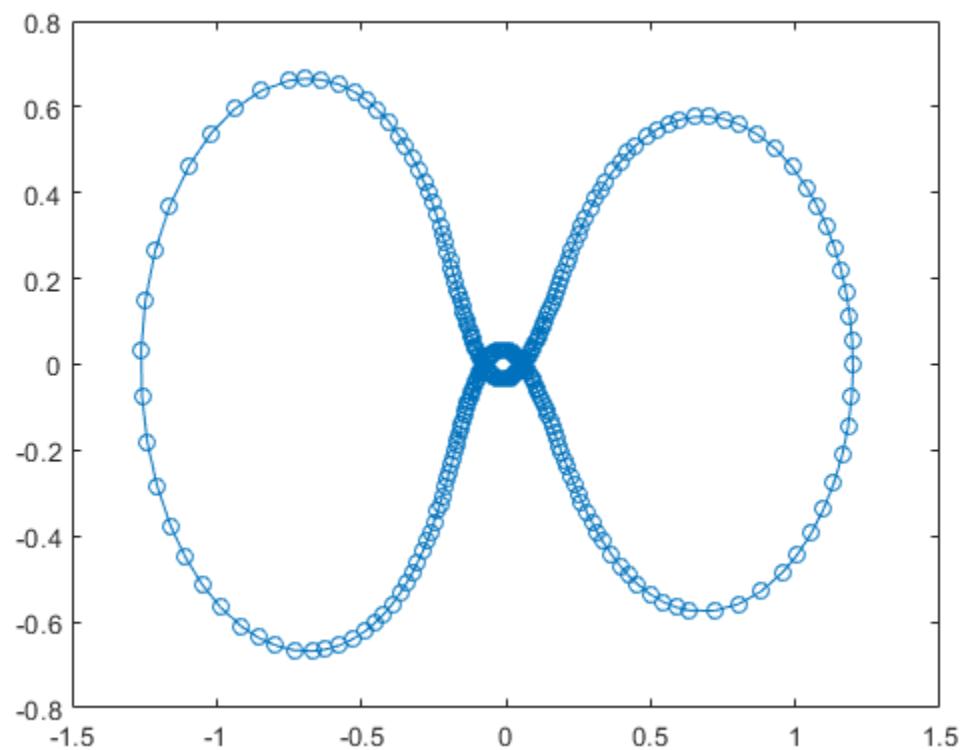
键入 **orbitode** 以运行该示例。

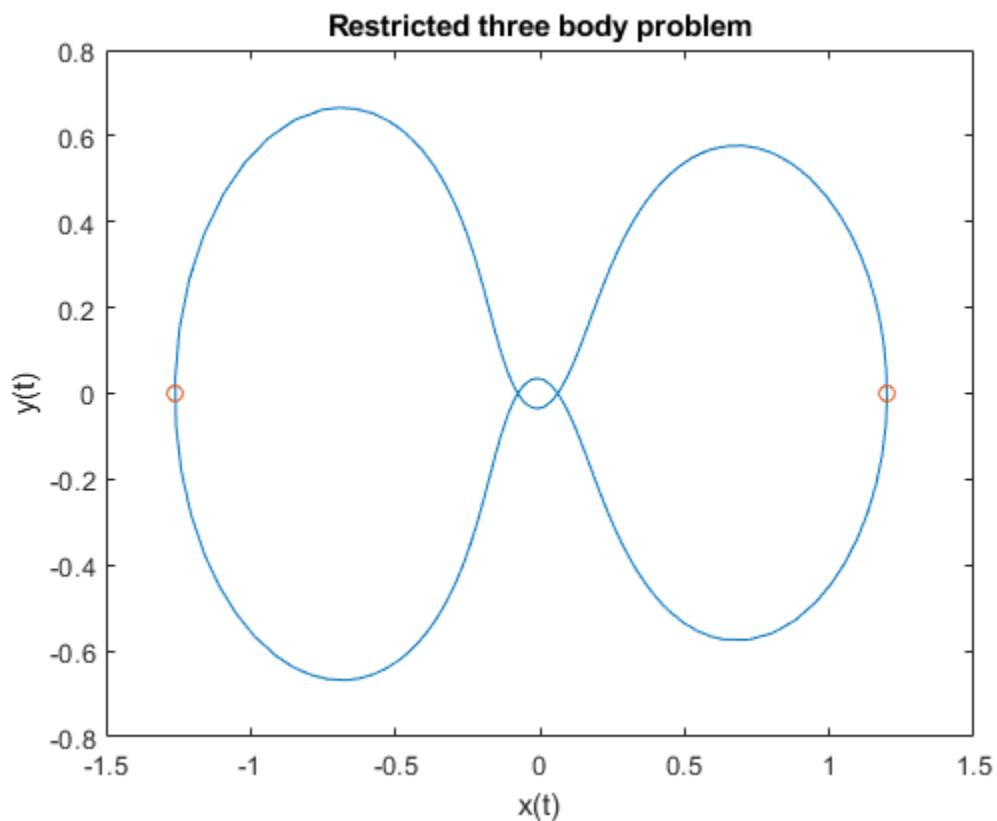
```
orbitode
```

This is an example of event location where the ability to specify the direction of the zero crossing is critical. Both the point of return to the initial point and the point of maximum distance have the same event function value, and the direction of the crossing is used to distinguish them.

Calling ODE45 with event functions active...

Note that the step sizes used by the integrator are NOT determined by the location of the events, and the events are still located accurately.





另请参阅

[odeget](#) | [odeset](#)

详细信息

- “选择 ODE 求解器” (第 11-2 页)
- “ODE 选项摘要” (第 11-9 页)
- “参数化函数” (第 10-2 页)

求解非刚性 ODE

本页包含两个使用 **ode45** 来求解非刚性常微分方程的示例。MATLAB® 拥有三个非刚性 ODE 求解器。

- **ode45**
- **ode23**
- **ode113**

对于大多数非刚性问题，**ode45** 的性能最佳。但对于允许较宽松的误差容限或刚度适中的问题，建议使用 **ode23**。同样，对于具有严格误差容限的问题，**ode113** 可能比 **ode45** 更加高效。

如果非刚性求解器需要很长时间才能解算问题或总是无法完成积分，则该问题可能是刚性问题。有关详细信息，请参阅“解算刚性 ODE”（第 11-21 页）。

示例：非刚性 van der Pol 方程

van der Pol 方程为二阶 ODE

$$y_1'' - \mu(1 - y_1^2)y_1' + y_1 = 0,$$

其中 $\mu > 0$ 为标量参数。通过执行 $y_1' = y_2$ 代换，将此方程重写为一阶 ODE 方程组。生成的一阶 ODE 方程组为

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \mu(1 - y_1^2)y_2 - y_1. \end{aligned}$$

ODE 方程组必须编码为 ODE 求解器能够使用的函数文件。ODE 函数的一般函数签名为

```
dydt = odefun(t,y)
```

即，函数必须同时接受 t 和 y 作为输入，即使它没有将 t 用于任何计算时亦如此。

函数文件 **vdp1.m** 使用 $\mu = 1$ 为该 van der Pol 方程编码。变量 y_1 和 y_2 表示为 $y(1)$ 和 $y(2)$ ，二元素列向量 $dydt$ 包含 y_1' 和 y_2' 的表达式。

```
function dydt = vdp1(t,y)
%VDP1 Evaluate the van der Pol ODEs for mu = 1
%
% See also ODE113, ODE23, ODE45.

% Jacek Kierzenka and Lawrence F. Shampine
% Copyright 1984-2014 The MathWorks, Inc.
```

```
dydt = [y(2); (1-y(1)^2)*y(2)-y(1)];
```

使用 **ode45** 函数、时间区间 **[0 20]** 和初始值 **[2 0]** 来解算该 ODE。输出为时间点列向量 t 和解数组 y 。 y 中的每一行都与 t 的相应行中返回的时间相对应。 y 的第一列与 y_1 相对应，第二列与 y_2 相对应。

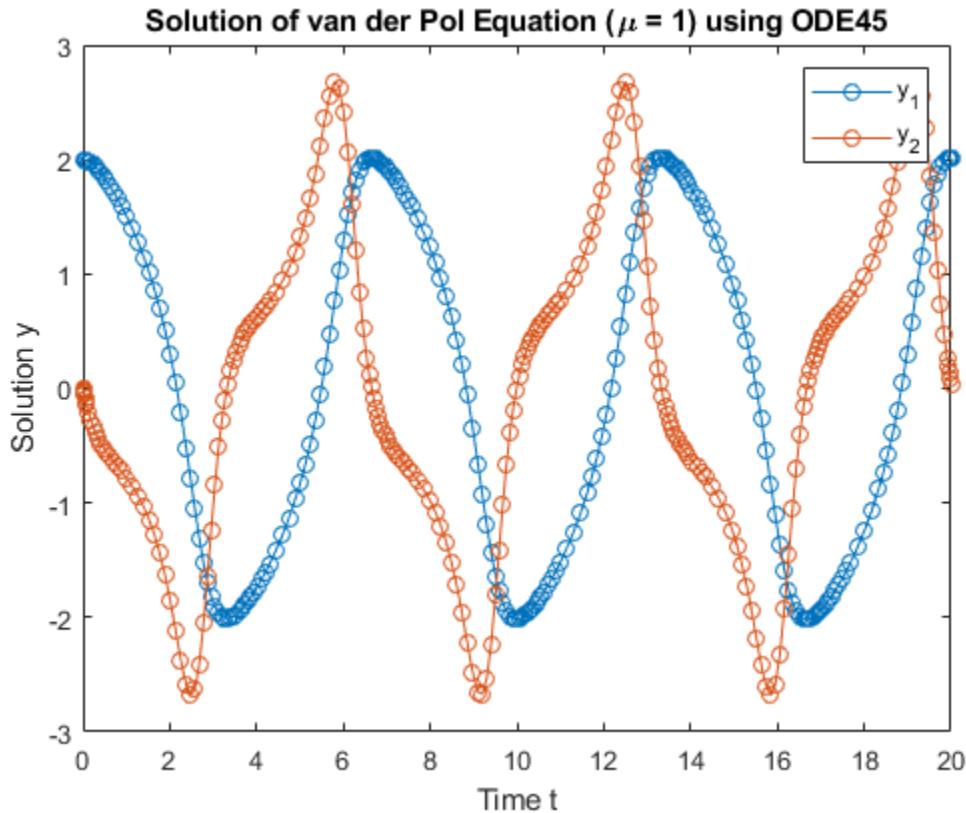
```
[t,y] = ode45(@vdp1,[0 20],[2; 0]);
```

绘制 y_1 和 y_2 的解对 t 的图。

```

plot(t,y(:,1),'-o',t,y(:,2),'-o')
title('Solution of van der Pol Equation (\mu = 1) using ODE45');
xlabel('Time t');
ylabel('Solution y');
legend('y_1','y_2')

```



vdpoode 函数可求解同一问题，但它接受的是用户指定的 μ 值。随着 μ 的增大，van der Pol 方程组将变成刚性。例如，对于值 $\mu = 1000$ ，您需要使用 **ode15s** 等刚性求解器来求解该方程组。

示例：非刚性欧拉方程

对于专用于非刚性问题的 ODE 求解器，不受外力作用的刚体对应的欧拉方程是其标准测试问题。

这些方程包括

$$\begin{aligned} y'_1 &= y_2 y_3 \\ y'_2 &= -y_1 y_3 \\ y'_3 &= -0.51 y_1 y_2. \end{aligned}$$

函数文件 **rigidode** 定义此一阶方程组，并在时间区间 [0 12] 上使用初始条件向量 [0; 1; 1]（该向量对应于 y_1 、 y_2 和 y_3 的初始值）对该方程组进行求解。局部函数 **f(t,y)** 用于编写该方程组的代码。

rigidode 在调用 **ode45** 时未使用任何输出参数，因此求解器会在每一步之后使用默认的输出函数 **odeplot** 自动绘制解点。

```
function rigidode
```

```
%RIGIDODE Euler equations of a rigid body without external forces.
% A standard test problem for non-stiff solvers proposed by Krogh. The
% analytical solutions are Jacobian elliptic functions, accessible in
% MATLAB. The interval here is about 1.5 periods; it is that for which
% solutions are plotted on p. 243 of Shampine and Gordon.
%
% L. F. Shampine and M. K. Gordon, Computer Solution of Ordinary
% Differential Equations, W.H. Freeman & Co., 1975.
%
% See also ODE45, ODE23, ODE113, FUNCTION_HANDLE.
%
% Mark W. Reichelt and Lawrence F. Shampine, 3-23-94, 4-19-94
% Copyright 1984-2014 The MathWorks, Inc.

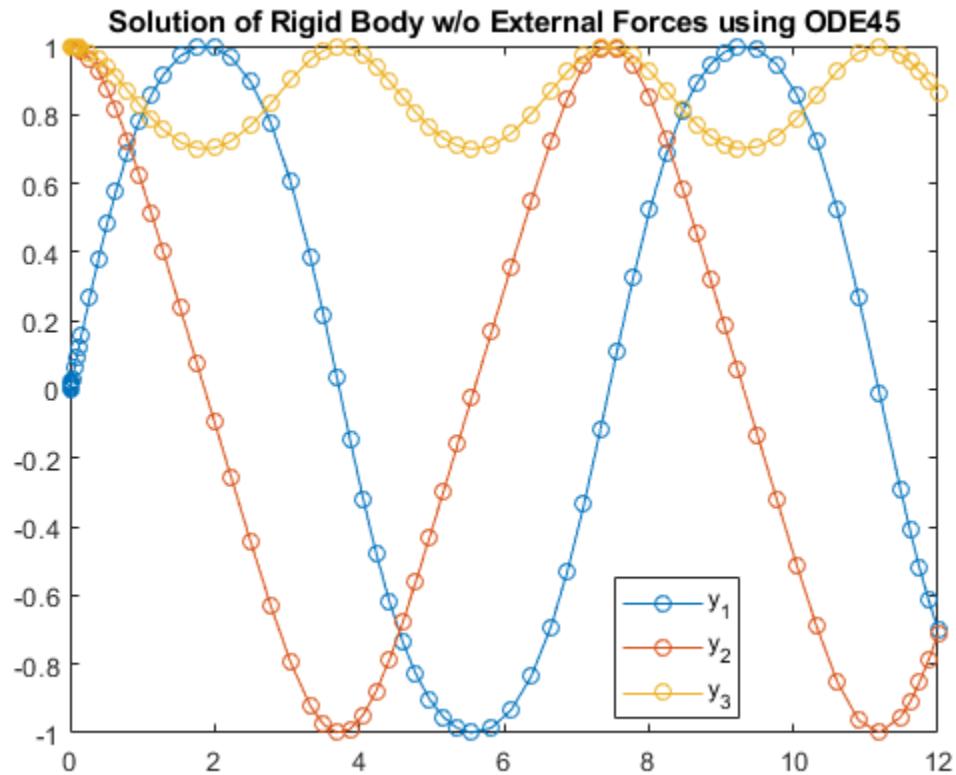
tspan = [0 12];
y0 = [0; 1; 1];

% solve the problem using ODE45
figure;
ode45(@f,tspan,y0);

% -----
function dydt = f(t,y)
dydt = [ -y(2)*y(3)
          -y(1)*y(3)
          -0.51*y(1)*y(2) ];
```

通过调用 `rigidode` 函数来解算非刚性欧拉方程。

```
rigidode
title('Solution of Rigid Body w/o External Forces using ODE45')
legend('y_1','y_2','y_3','Location','Best')
```



另请参阅

[ode113](#) | [ode23](#) | [ode45](#)

详细信息

- “选择 ODE 求解器” (第 11-2 页)
- “参数化函数” (第 10-2 页)

解算刚性 ODE

本页包含两个使用 **ode15s** 解算刚性常微分方程的示例。MATLAB® 拥有四个专用于刚性 ODE 的求解器。

- **ode15s**
- **ode23s**
- **ode23t**
- **ode23tb**

对于大多数刚性问题，**ode15s** 的性能最佳。但如果问题允许较宽松的误差容限，则 **ode23s**、**ode23t** 和 **ode23tb** 可能更加高效。

什么是刚性 ODE?

对于一些 ODE 问题，求解器采用的步长被强制缩小为与积分区间相比过小的级别，甚至在解曲线平滑的区域亦如此。这些步长可能过小，以至于遍历很短的时间区间都可能需要数百万次计算。这可能导致求解器积分失败，即使积分成功也需要花费很长时间。

导致 ODE 求解器出现此行为的方程称为刚性方程。刚性 ODE 造成的问题是，显式求解器（例如 **ode45**）获取解的速度慢得令人无法忍受。这是将 **ode45** 与 **ode23** 和 **ode113** 一同归类为非刚性求解器的原因所在。

专用于刚性 ODE 的求解器称为刚性求解器，它们通常在每一步中完成更多的计算工作。这样做的好处是，它们能够采用大得多的步长，并且与非刚性求解器相比提高了数值稳定性。

求解器选项

对于刚性问题，使用 **odeset** 指定 Jacobian 矩阵尤为重要。刚性求解器使用 Jacobian 矩阵 $\frac{\partial f_i}{\partial y_j}$ 来预测 ODE 在积分过程中的局部行为，因此提供 Jacobian 矩阵（或者对于大型稀疏方程组提供其稀疏模式）对于提高效率和可靠性而言至关重要。使用 **odeset** 的 **Jacobian**、**JPattern** 或 **Vectorized** 选项来指定 Jacobian 的相关信息。如果没有提供 Jacobian，则求解器将使用有限差分对其进行数值预测。

有关其他求解器选项的完整列表，请参阅 **odeset**。

示例：刚性 van der Pol 方程

van der Pol 方程为二阶 ODE

$$y_1'' - \mu(1 - y_1^2)y_1' + y_1 = 0,$$

其中 $\mu > 0$ 为标量参数。当 $\mu = 1$ 时，生成的 ODE 方程组为非刚性方程组，可以使用 **ode45** 轻松求解。但如果将 μ 增大至 1000，则解会发生显著变化，并会在明显更长的时间段中显示振荡。求初始值问题的近似解变得更加复杂。由于此特定问题是刚性问题，因此专用于非刚性问题的求解器（如 **ode45**）的效率非常低下且不切实际。针对此问题应改用 **ode15s** 等刚性求解器。

通过执行代换 $y'_1 = y_2$ ，将该 van der Pol 方程重写为一阶 ODE 方程组。生成的一阶 ODE 方程组为

$$\begin{aligned} y'_1 &= y_2 \\ y'_2 &= \mu(1 - y_1^2)y_2 - y_1. \end{aligned}$$

vdp1000 函数使用 $\mu = 1000$ 计算 van der Pol 方程。

```

function dydt = vdp1000(t,y)
%VDP1000 Evaluate the van der Pol ODEs for mu = 1000.
%
% See also ODE15S, ODE23S, ODE23T, ODE23TB.
%
% Jacek Kierzenka and Lawrence F. Shampine
% Copyright 1984-2014 The MathWorks, Inc.

dydt = [y(2); 1000*(1-y(1)^2)*y(2)-y(1)];

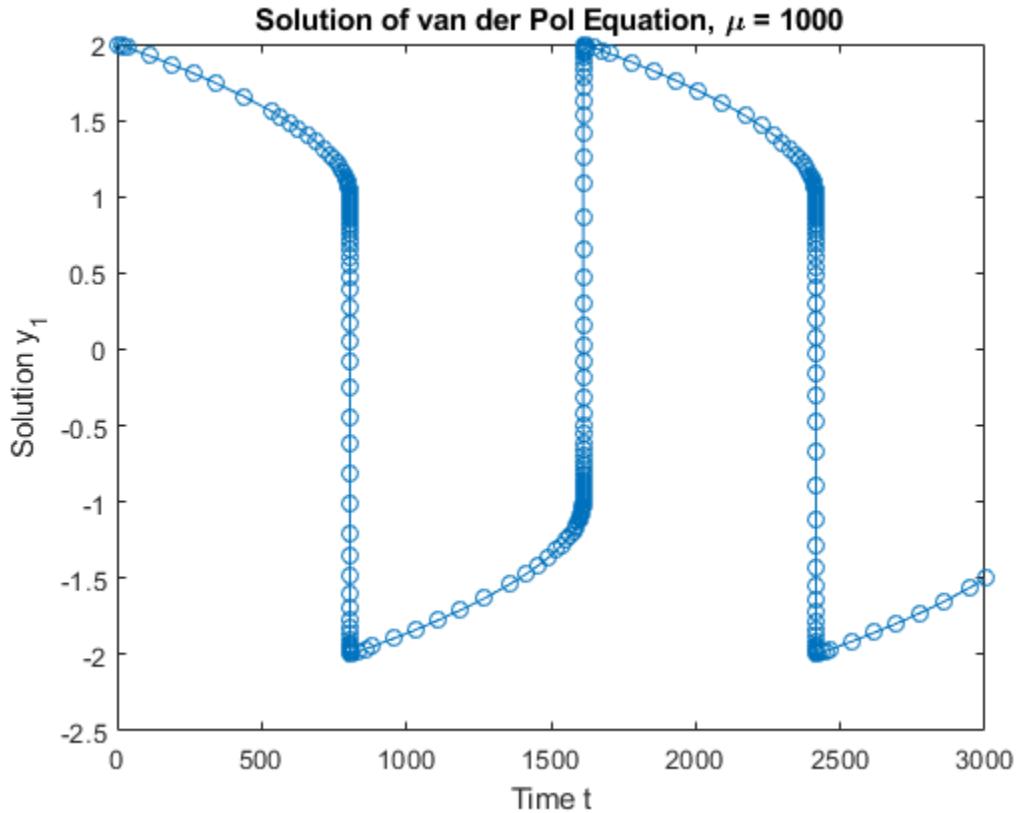
```

使用 `ode15s` 函数和初始条件向量 `[2; 0]`，在时间区间 `[0 3000]` 上解算此问题。由于是标量，因此仅绘制解的第一个分量。

```

[t,y] = ode15s(@vdp1000,[0 3000],[2; 0]);
plot(t,y(:,1),'-o');
title('Solution of van der Pol Equation, \mu = 1000');
xlabel('Time t');
ylabel('Solution y_1');

```



`vdpode` 函数也可以求解同一问题，但它接受的是用户指定的 μ 值。随着 μ 的增大，该方程组的刚性逐渐增强。

示例：稀疏 Brusselator 方程组

经典 Brusselator 方程组可能为大型刚性稀疏矩阵。Brusselator 方程组可模拟化学反应中的扩算，并表示为涉及 u 、 v 、 u' 和 v' 的方程组。

$$\begin{aligned} u'_i &= 1 + u_i^2 v_i - 4u_i + \alpha (N+1)^2 (u_{i-1} - 2u_i + u_{i+1}) \\ v'_i &= 3u_i - u_i^2 v_i + \alpha (N+1)^2 (v_{i-1} - 2v_i + v_{i+1}) \end{aligned}$$

函数文件 **brussode** 使用 $\alpha = 1/50$ 在时间区间 $[0,10]$ 上对这组方程进行求解。初始条件为

$$\begin{aligned} u_j(0) &= 1 + \sin(2\pi x_j) \\ v_j(0) &= 3, \end{aligned}$$

其中, 当 $i = 1, \dots, N$ 时, $x_j = i/N + 1$ 。因此, 该方程组中有 $2N$ 个方程, 但如果这些方程按 $u_1, v_1, u_2, v_2, \dots$ 的形式排序, 则 Jacobian $\partial f / \partial y$ 为具有常量宽度 5 的带状矩阵。随着 N 的增大, 此问题的刚性逐渐增强, 并且 Jacobian 矩阵的稀疏性也逐渐增大。

函数调用 **brussode(N)** (其中 $N \geq 2$) 为方程组中的 N (对应于网格点数量) 指定值。默认情况下, **brussode** 使用 $N = 20$ 。

brussode 包含一些子函数:

- 嵌套函数 **f(t,y)** 用于编写 Brusselator 问题的方程组代码, 并返回一个向量。
- 局部函数 **jpattern(N)** 返回由 1 和 0 组成的稀疏矩阵, 从而显示 Jacobian 矩阵中非零值的位置。此矩阵将赋给 options 结构体的 **JPattern** 字段。ODE 求解器使用此稀疏模式, 生成稀疏矩阵形式的 Jacobian 数值矩阵。在问题中提供此稀疏模式可将生成 $2N \times 2N$ Jacobian 矩阵所需的函数计算量从 $2N$ 次大幅减少至仅仅 4 次。

```
function brussode(N)
%BRUSSODE Stiff problem modelling a chemical reaction (the Brusselator).
% The parameter N >= 2 is used to specify the number of grid points; the
% resulting system consists of 2N equations. By default, N is 20. The
% problem becomes increasingly stiff and increasingly sparse as N is
% increased. The Jacobian for this problem is a sparse constant matrix
% (banded with bandwidth 5).
%
% The property 'JPattern' is used to provide the solver with a sparse
% matrix of 1's and 0's showing the locations of nonzeros in the Jacobian
% df/dy. By default, the stiff solvers of the ODE Suite generate Jacobians
% numerically as full matrices. However, when a sparsity pattern is
% provided, the solver uses it to generate the Jacobian numerically as a
% sparse matrix. Providing a sparsity pattern can significantly reduce the
% number of function evaluations required to generate the Jacobian and can
% accelerate integration. For the BRUSSODE problem, only 4 evaluations of
% the function are needed to compute the 2N x 2N Jacobian matrix.
%
% Setting the 'Vectorized' property indicates the function f is
% vectorized.
%
% E. Hairer and G. Wanner, Solving Ordinary Differential Equations II,
% Stiff and Differential-Algebraic Problems, Springer-Verlag, Berlin,
% 1991, pp. 5-8.
%
% See also ODE15S, ODE23S, ODE23T, ODE23TB, ODESET, FUNCTION_HANDLE.
%
% Mark W. Reichelt and Lawrence F. Shampine, 8-30-94
% Copyright 1984-2014 The MathWorks, Inc.
```

```
% Problem parameter, shared with the nested function.
if nargin<1
    N = 20;
end

tspan = [0; 10];
y0 = [1+sin((2*pi/(N+1))*(1:N)); repmat(3,1,N)];
options = odeset('Vectorized','on','JPattern',jpattern(N));

[t,y] = ode15s(@f,tspan,y0,options);

u = y(:,1:2:end);
x = (1:N)/(N+1);
figure;
surf(x,t,u);
view(-40,30);
xlabel('space');
ylabel('time');
zlabel('solution u');
title(['The Brusselator for N = ' num2str(N)]);

% -----
% Nested function -- N is provided by the outer function.
%

function dydt = f(t,y)
    % Derivative function
    c = 0.02 * (N+1)^2;
    dydt = zeros(2*N,size(y,2));    % preallocate dy/dt

    % Evaluate the 2 components of the function at one edge of the grid
    % (with edge conditions).
    i = 1;
    dydt(i,:) = 1 + y(i+1,:).*y(i,:).^2 - 4*y(i,:)^2 + c*(1-2*y(i,:)+y(i+2,:));
    dydt(i+1,:) = 3*y(i,:)^2 - y(i+1,:).*y(i,:)^2 + c*(3-2*y(i+1,:)+y(i+3,:));

    % Evaluate the 2 components of the function at all interior grid points.
    i = 3:2:N-3;
    dydt(i,:) = 1 + y(i+1,:).*y(i,:)^2 - 4*y(i,:)^2 + ...
        c*(y(i-2,:)-2*y(i,:)+y(i+2,:));
    dydt(i+1,:) = 3*y(i,:)^2 - y(i+1,:).*y(i,:)^2 + ...
        c*(y(i-1,:)-2*y(i+1,:)+y(i+3,:));

    % Evaluate the 2 components of the function at the other edge of the grid
    % (with edge conditions).
    i = 2*N-1;
    dydt(i,:) = 1 + y(i+1,:).*y(i,:)^2 - 4*y(i,:)^2 + c*(y(i-2,:)-2*y(i,:)+1);
    dydt(i+1,:) = 3*y(i,:)^2 - y(i+1,:).*y(i,:)^2 + c*(y(i-1,:)-2*y(i+1,:)+3);
end
% -----
end % brussode

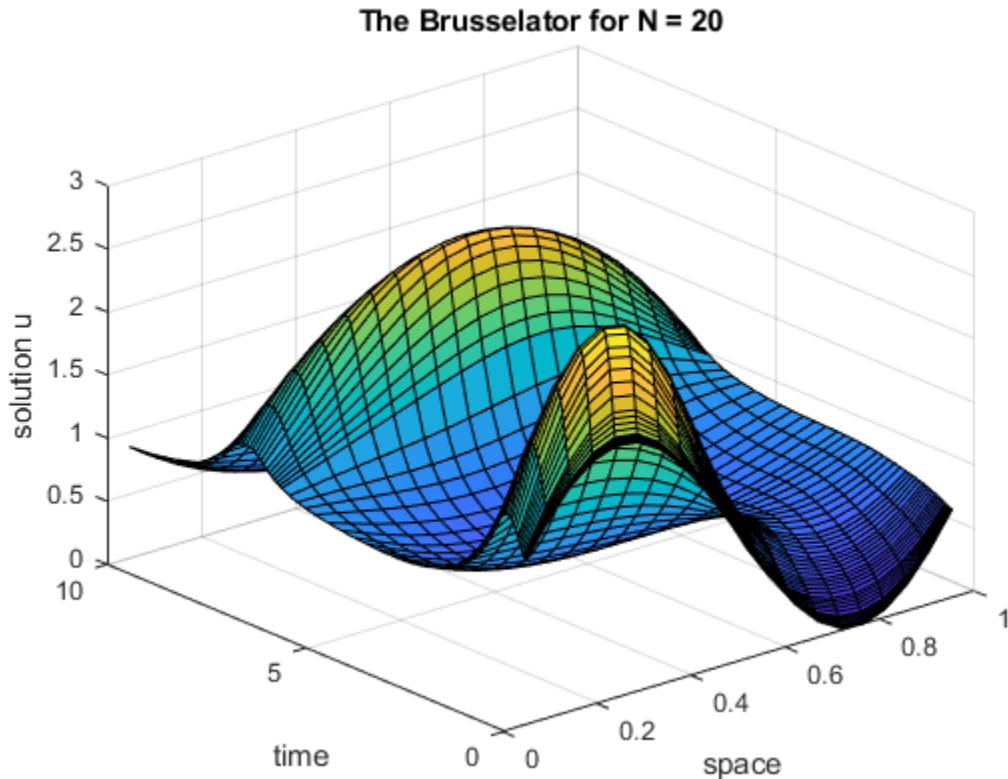
% -----
% Subfunction -- the sparsity pattern
%
```

```

function S = jpattern(N)
% Jacobian sparsity pattern
B = ones(2*N,5);
B(2:2:2*N,2) = zeros(N,1);
B(1:2:2*N-1,4) = zeros(N,1);
S = spdiags(B,-2:2,2*N,2*N);
end
%
```

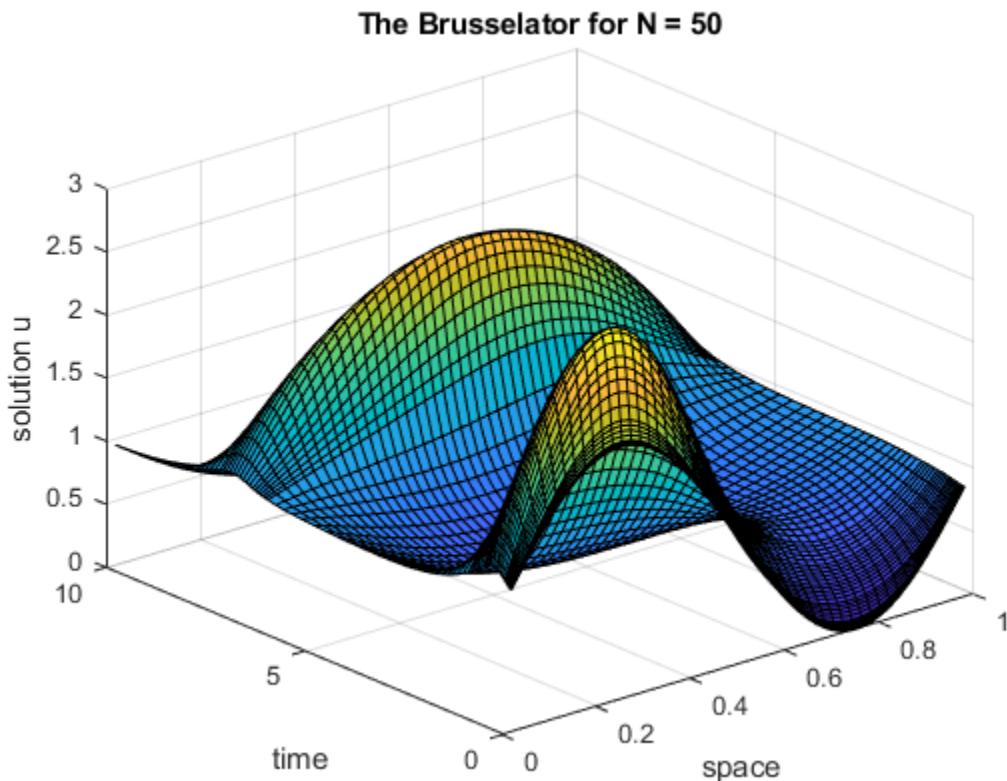
通过运行函数 **brussode**, 对 $N = 20$ 时的 Brusselator 方程组求解。

brussode



通过为 **brussode** 指定输入, 对 $N = 50$ 时的方程组求解。

brussode(50)



另请参阅

[ode15s](#) | [ode23s](#) | [ode23t](#) | [ode23tb](#)

详细信息

- “选择 ODE 求解器” (第 11-2 页)
- “ODE 选项摘要” (第 11-9 页)
- “参数化函数” (第 10-2 页)

解算微分代数方程 (DAE)

本节内容

- “什么是微分代数方程？”（第 11-27 页）
- “一致的初始条件”（第 11-28 页）
- “微分指数”（第 11-28 页）
- “施加非负性”（第 11-28 页）
- “将 Robertson 问题作为半隐式微分代数方程 (DAE) 求解”（第 11-29 页）

什么是微分代数方程？

微分代数方程是一类微分方程，其中一个或多个因变量导数未出现在方程中。方程中出现的未包含其导数的变量称为代数变量，代数变量的存在意味着您不能将这些方程记为显式形式 $y' = f(t, y)$ 。相反，您可以解算下列形式的 DAE：

- **ode15s** 和 **ode23t** 求解器可以使用奇异质量矩阵 $M(t, y)y' = f(t, y)$ 来解算微分指数为 1 的线性隐式问题，包括以下形式的半显式 DAE

$$\begin{aligned} y' &= f(t, y, z) \\ 0 &= g(t, y, z). \end{aligned}$$

在此形式中，由于主对角线存在一个或多个零值，因此代数变量的存在会产生奇异质量矩阵。

$$My = \begin{pmatrix} y'_1 & 0 & \cdots & 0 \\ 0 & y'_2 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & 0 \end{pmatrix}.$$

默认情况下，求解器会自动检验质量矩阵的奇异性，以检测 DAE 方程组。如果您提前知道奇异性，则可将 **odeset** 的 **MassSingular** 选项设为 'yes'。对于 DAE，您还可以使用 **odeset** 的 **InitialSlope** 属性为求解器提供 y'_0 的初始条件估计值。除此之外，还可在调用求解器时指定 y_0 的常用初始条件。

- **ode15i** 求解器可解算更通用的完全隐式形式的 DAE

$$f(t, y, y') = 0.$$

在完全隐式形式下，代数变量的存在会产生奇异 Jacobian 矩阵。这是因为，由于至少有一个变量的导数没有出现在方程中，因此矩阵中的对应列必定全部为零值。

$$J = \frac{\partial f}{\partial y} = \begin{pmatrix} \frac{\partial f_1}{\partial y'_1} & \cdots & \frac{\partial f_1}{\partial y'_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial y'_1} & \cdots & \frac{\partial f_m}{\partial y'_n} \end{pmatrix}$$

ode15i 求解器要求您同时为 y'_0 和 y_0 指定初始条件。此外，与其他 ODE 求解器不同，**ode15i** 要求为方程编码的函数能够接受额外输入：**odefun(t,y,yp)**。

DAE 会产生各种方程组，因为物理守恒定律通常具有类似 $x + y + z = 0$ 这样的形式。如果已在方程中显式定义 x 、 x' 、 y 和 y' ，则此守恒方程无需 z' 表达式便足以解算 z 。

一致的初始条件

在解算 DAE 时，可以同时为 y'_0 和 y_0 指定初始条件。**ode15i** 求解器要求同时将这两个初始条件指定为输入参数。对于 **ode15s** 和 **ode23t** 求解器， y'_0 的初始条件是可选的（但可使用 **odeset** 的 **InitialSlope** 选项指定）。这两种情况下，您所指定的初始条件可能与正在尝试解算的方程不相符。彼此冲突的初始条件称为不一致。初始条件的处理因求解器而异：

- **ode15s** 和 **ode23t** - 如果您没有为 y'_0 指定初始值，则求解器会自动基于您为 y_0 提供的初始条件计算一致的初始条件。如果您为 y'_0 指定了不一致的初始条件，则求解器会将这些值作为估计值进行处理，尝试计算接近估计值的一致值，并继续解算该问题。
- **ode15i** - 您为求解器提供的初始条件必须一致，并且 **ode15i** 不会检查所提供的值的一致性。辅助函数 **decic** 可计算满足这一要求的一致初始条件。

微分指数

DAE 的特征是其作为奇异性度量的微分指数。通过对方程进行微分，可以消除代数变量，并且如果执行此操作的次数足够多，这些方程将呈现为显式 ODE 方程组。DAE 方程组的微分指数是为了将方程组表示为等效的显式 ODE 方程组必须执行的求导次数。因此，ODE 的微分指数为 0。

微分指数为 1 的 DAE 示例如下

$$y(t) = k(t).$$

对此方程，只需执行一次求导便可获得显式 ODE 形式

$$y' = k'(t).$$

微分指数为 2 的 DAE 示例如下

$$y'_1 = y_2$$

$$0 = k(t) - y_1.$$

这些方程要求进行两次求导才能重写为显式 ODE 形式

$$y'_1 = k'(t)$$

$$y''_2 = k''(t).$$

ode15s 和 **ode23t** 求解器仅可解算微分指数为 1 的 DAE。如果您的方程微分指数为 2 或更高，则需要将方程重写为微分指数为 1 的等效 DAE 方程组。您可随时对 DAE 方程组求导并将其重写为微分指数为 1 的等效 DAE 方程组。请注意，如果您将代数方程替换为其导数，则可能已删除某些约束。如果这些方程不再包含原始约束，则数值解可能发生漂移。

如果您有 Symbolic Math Toolbox，则请参阅 “Solve Differential Algebraic Equations (DAEs)” (Symbolic Math Toolbox) 以获取详细信息。

施加非负性

odeset (第 11-9 页) 的大多数选项与 DAE 求解器 **ode15s**、**ode23t** 和 **ode15i** 一起使用时能按预期工作。然而，一个明显的例外是使用 **NonNegative** (第 11-32 页) 选项。**NonNegative** 选项不支持应用于具有质量矩阵的问题的隐式求解器 (**ode15s**、**ode23t**、**ode23tb**)。因此，您不能使用此选项对 DAE 问题施加非负性约束，DAE 问题一定有奇异质量矩阵。有关详细信息，请参阅 [1] (第 11-30 页)。

将 Robertson 问题作为半隐式微分代数方程 (DAE) 求解

此示例将 ODE 方程组重新表示为微分代数方程组 (DAE)。hb1ode.m 中的 Robertson 问题是刚性 ODE 解算程序的经典测试问题。方程组为：

$$\begin{aligned}y'_1 &= -0.04y_1 + 10^4y_2y_3 \\y'_2 &= 0.04y_1 - 10^4y_2y_3 - (3 \times 10^7)y_2^2 \\y'_3 &= (3 \times 10^7)y_2^2.\end{aligned}$$

hb1ode 求此 ODE 方程组的稳态，初始条件为 $y_1 = 1$ 、 $y_2 = 0$ 和 $y_3 = 0$ 。但这些方程也满足线性守恒定律，

$$y'_1 + y'_2 + y'_3 = 0.$$

在解和初始条件方面，守恒定律为

$$y_1 + y_2 + y_3 = 1.$$

通过使用守恒定律确定 y_3 的状态，该方程组可以重写为 DAE 方程组。这会将问题重新表示为 DAE 方程组

$$\begin{aligned}y'_1 &= -0.04y_1 + 10^4y_2y_3 \\y'_2 &= 0.04y_1 - 10^4y_2y_3 - (3 \times 10^7)y_2^2 \\0 &= y_1 + y_2 + y_3 - 1.\end{aligned}$$

此方程组的微分指数为 1，因为只需 y_3 的一个导数就能使其成为 ODE 方程组。因此，在解算该方程组之前，不需要进行更多变换。

函数 **robertsdae** 为此 DAE 方程组编码。将 **robertsdae.m** 保存在您的当前文件夹中，以运行该示例。

```
function out = robertsdae(t,y)
out = [-0.04*y(1) + 1e4*y(2).*y(3)
       0.04*y(1) - 1e4*y(2).*y(3) - 3e7*y(2).^2
       y(1) + y(2) + y(3) - 1];
```

hb1dae.m 中提供了用这种方法表示 Robertson 问题的完整示例代码。

使用 **ode15s** 解算 DAE 方程组。根据守恒定律，显然需要一致的 **y0** 初始条件。使用 **odeset** 设置选项：

- 使用常量质量矩阵表示方程组的左侧。

$$\begin{pmatrix} y'_1 \\ y'_2 \\ 0 \end{pmatrix} = My' \rightarrow M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

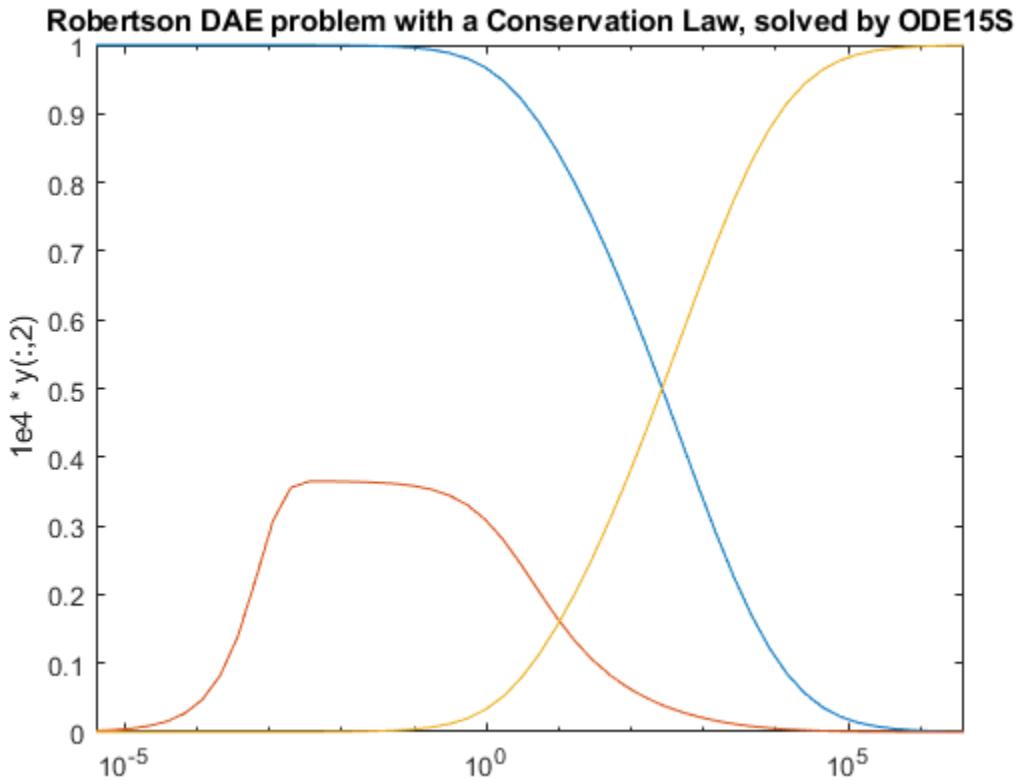
- 将相对误差容限设为 **1e-4**。
- 使用 **1e-10** 的绝对误差作为第二个解分量，因为标度范围与其他分量相差很大。
- 将 '**MassSingular**' 选项保留其默认值 '**maybe**'，以测试 DAE 的自动检测。

```
y0 = [1; 0; 0];
tspan = [0 4*logspace(-6,6)];
M = [1 0 0; 0 1 0; 0 0 0];
```

```
options = odeset('Mass',M,'RelTol',1e-4,'AbsTol',[1e-6 1e-10 1e-6]);
[t,y] = ode15s(@robertsdae,tspan,y0,options);
```

对解绘图。

```
y(:,2) = 1e4*y(:,2);
semilogx(t,y);
ylabel('1e4 * y(:,2)');
title('Robertson DAE problem with a Conservation Law, solved by ODE15S');
```



参考

[1] Shampine, L.F., S. Thompson, J.A. Kierzenka, and G.D. Byrne. "Non-negative solutions of ODEs." *Applied Mathematics and Computation*. Vol. 170, 2005, pp. 556-569.

另请参阅

[ode15i](#) | [ode15s](#) | [ode23t](#) | [odeset](#)

详细信息

- “选择 ODE 求解器” (第 11-2 页)
- “ODE 选项摘要” (第 11-9 页)
- “Equation Solving” (Symbolic Math Toolbox)

外部网站

- 在 MATLAB 和 Simulink 中解算微分指数为 1 的 DAE

非负 ODE 解

本主题说明如何将 ODE 解约束为非负解。施加非负约束不一定总是可有可无，在某些情况下，由于方程的物理解释或解性质的原因，可能有必要施加非负约束。仅在必要时对解施加此约束，例如不这样做积分就会失败或者解将不适用的情况。

如果解的特定分量必须为非负，则使用 `odeset` 来设置这些分量的索引的 `NonNegative` 选项。此选项不适用于 `ode23s`、`ode15i`，也不适用于用来求解涉及质量矩阵的问题的隐式求解器（`ode15s`、`ode23t`、`ode23tb`）。特别是，不能对 DAE 问题施加非负性约束，DAE 问题一定有奇异质量矩阵。

示例：绝对值函数

考虑初始值问题

$$y' = -|y|,$$

该问题使用初始条件 $y(0) = 1$ 在区间 $[0, 40]$ 上求解。此 ODE 的解将衰减到零。如果求解器生成负解值，则它会开始通过此值来跟踪 ODE 的解，随着计算得出的解逐渐发散为 $-\infty$ ，计算最终会失败。使用 `NonNegative` 选项可防止此积分失败。

将 $y(t) = e^{-t}$ 的解析解分别与使用不带额外选项的 `ode45` 得出的 ODE 解和设定 `NonNegative` 选项时得出的 ODE 解进行比较。

```
ode = @(t,y) -abs(y);

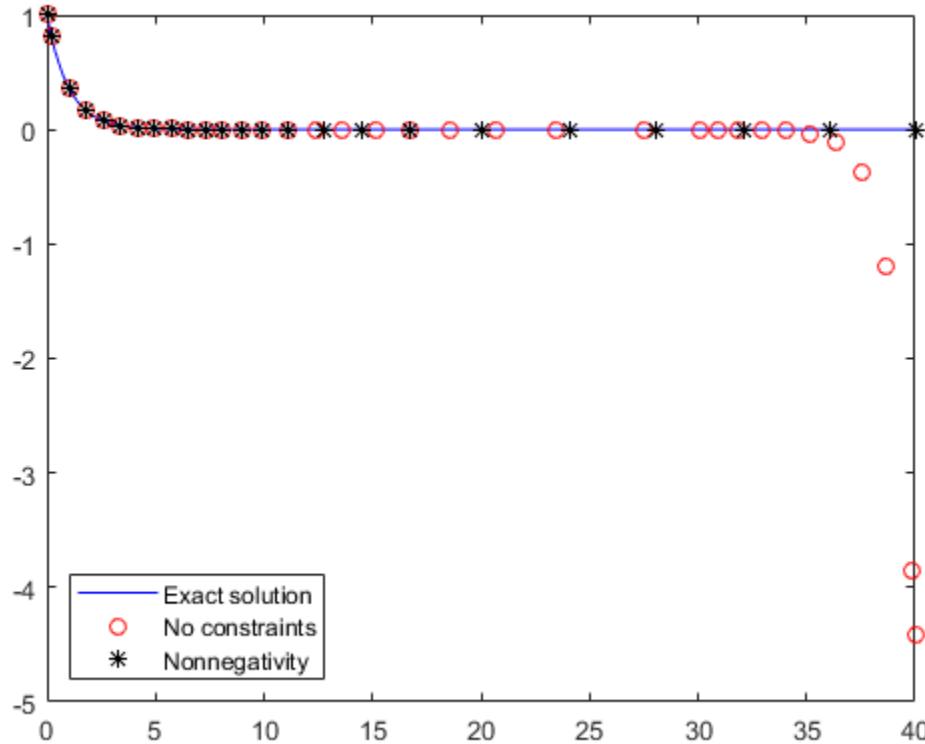
% Standard solution with |ode45|
options1 = odeset('Refine',1);
[t0,y0] = ode45(ode,[0 40],1,options1);

% Solution with nonnegative constraint
options2 = odeset(options1,'NonNegative',1);
[t1,y1] = ode45(ode,[0 40],1,options2);

% Analytic solution
t = linspace(0,40,1000);
y = exp(-t);
```

绘制这三个解进行比较。施加非负约束对于防止解向 $-\infty$ 发展至关重要。

```
plot(t,y,'b-',t0,y0,'ro',t1,y1,'k*');
legend('Exact solution','No constraints','Nonnegativity',...
    'Location','SouthWest')
```



示例：膝盖问题

另一个要求非负解的问题示例是在示例文件 `kneeode` 中编码的膝盖问题。方程是：

$$\epsilon y' = (1-x)y - y^2,$$

该问题使用初始条件 $y(0) = 1$ 在区间 $[0, 2]$ 上求解。通常采用参数 ϵ 以满足 $0 < \epsilon \ll 1$ ，并且此问题使用 $\epsilon = 1 \times 10^{-6}$ 。此 ODE 的解在 $x < 1$ 时趋近于 $y = 1 - x$ ，在 $x > 1$ 时趋近于 $y = 0$ 。但通过使用默认容差计算数值解可以看到，解在整个积分区间中遵循 $y = 1 - x$ 等倾线。施加非负约束会得到正确的解。

在使用和不使用非负值约束两种条件下解算膝盖问题。

```

epsilon = 1e-6;
y0 = 1;
xspan = [0 2];
odefcn = @(x,y,epsilon) ((1-x)*y - y^2)/epsilon;

% Solve without imposing constraints
[x1,y1] = ode15s(@(x,y) odefcn(x,y,epsilon), xspan, y0);

% Impose a nonnegativity constraint
options = odeset('NonNegative',1);
[x2,y2] = ode15s(@(x,y) odefcn(x,y,epsilon), xspan, y0, options);

```

绘制解进行比较。

```

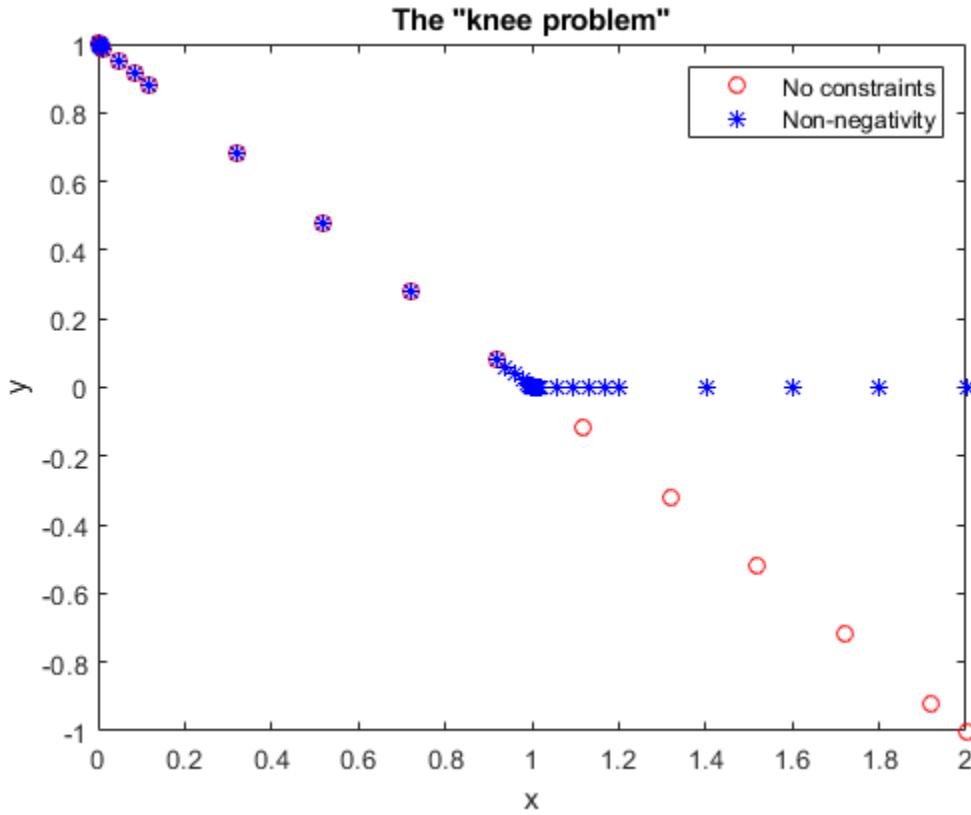
plot(x1,y1,'ro',x2,y2,'b*')
axis([0,2,-1,1])

```

```

title("The \"knee problem\")
legend('No constraints','Non-negativity')
xlabel('x')
ylabel('y')

```



参考

[1] Shampine, L.F., S. Thompson, J.A. Kierzenka, and G.D. Byrne, "Non-negative solutions of ODEs," Applied Mathematics and Computation Vol. 170, 2005, pp. 556-569.

另请参阅

[odeset](#)

详细信息

- “选择 ODE 求解器” (第 11-2 页)
- “ODE 选项摘要” (第 11-9 页)

常见 ODE 问题疑难解答

误差容限

疑问或问题	回答
如何选择误差阈值 RelTol 和 AbsTol ?	<p>RelTol 为相对精度容差，用于控制计算答案中的正确位数。AbsTol 为绝对误差容限，用于控制计算答案与实际解之间的差异。在每个步长中，解分量 i 中的误差 e 将满足</p> $ e(i) \leq \max(\text{RelTol} * \text{abs}(y(i)), \text{AbsTol}(i))$ <p>粗略地讲，这意味着您希望 RelTol 更正所有解分量中的位数（但小于阈值 AbsTol(i) 的解分量除外）。当分量 $y(i)$ 非常小时，即使您对此分量不感兴趣，您仍必须指定足够小的 AbsTol(i) 值，以便获取 $y(i)$ 中的某些正确位数，从而准确计算您感兴趣的分量。</p>
我希望求得达到计算机精度的答案。为什么不能直接将 RelTol 设为 eps ?	可以接近计算机精度，但不能如此接近。求解器不允许 RelTol 接近 eps ，因为它们会尝试求连续函数的近似值。当容差与 eps 相当时，计算机的计算会导致所有函数看似不连续。
我如何通知求解器我并不在乎如何获取某个解分量的正确答案？	可以增大与此解分量相对应的绝对误差容限 AbsTol 。如果容差大于此解分量，这表明不需要更正该分量中的任何位数。求解器可能必须获取此分量中的某些正确位数才能准确计算其他分量，但它通常会自动处理此过程。

问题规模

疑问或问题	回答
使用 ODE 套件能够解算多大的问题?	<p>主要限制在于内存和时间。在每个时间步长中，适用于非刚性问题的求解器分配长度为 n 的向量，其中 n 为方程组中的方程数。适用于刚性问题的求解器分配长度为 n 的向量，但还会分配一个 $n \times n$ Jacobian 矩阵。对于这些求解器而言，使用 odeset 的 JPattern 选项来指定 Jacobian 稀疏模式可能更为有利。</p> <p>如果问题是非刚性问题，或者使用的是 JPattern 选项，则可以解算具有上千个未知数的问题。但是，此种情况下，存储结果可能会带来问题。要求求解器仅在特定点求解，或者调用没有任何输出参数的求解器并使用输出函数来监控解。</p>

疑问或问题	回答
我解算的方程组非常大，但是我仅关心其中几个 y 分量。是否有任何方法可避免存储所有元素？	有。 OutputFcn 选项专门用于此用途。当调用没有输出参数的求解器时，求解器不会分配存储来保存所有历史解，而会在每个时间步长中调用 OutputFcn(t,y,flag) 。要保留特定元素的历史解，请编写用于仅存储或绘制所关注的元素的输出函数。
启动积分有哪些成本，如何减少启动成本？	当求解器尝试查找适用于问题标量的步长大小时，会带来最大的启动成本。如果您碰巧知道适当的步长，请使用 InitialStep 选项。例如，如果您在事件查找循环中重复调用积分器，则在此事件之前采用的最后一个步长可能已正确缩放，以用于后续积分。键入 <code>edit ballode</code> 以查看示例。
积分器采用的第一个步长太大，并且错过重要行为。	使用 InitialStep 选项可以指定第一个步长。积分器尝试使用此步长，然后在必要时缩小它。

解分量

疑问或问题	回答
解似乎与我的预期不符。	<p>如果您的预期是正确的，则从默认值开始减小误差容限。需要较小的相对误差容限才能正确解算在“较长”区间积分的问题，以及不稳定性适中的问题。</p> <p>检查是否存在在某段时间内小于其绝对误差容限的解分量。如果是这种情况，则表明您对这些分量中的正确位数未做任何要求。对于这些分量而言，这可能是可接受的，但无法精确计算这些分量可能会降低依赖于这些分量的其他分量的精度。</p>
我的绘图不够平滑。	<p>从默认值开始，增大 Refine 的值，在 <code>ode45</code> 中此默认值为 4，在其他求解器中此默认值为 1。 Refine 的值越大，求解器生成的输出点越多。 Refine 值对执行速度影响甚微。</p>
我在计算解时绘制解，此解看似正常，但代码在某个点出现问题。	<p>首先验证 ODE 函数在出现问题的位置附近是否平滑。如果不平滑，则求解器必须采用较小步长才能解决此问题。将积分区间分解为多个片段，使 ODE 函数在其上显示为平滑函数，可能会有帮助。</p> <p>如果此函数非常平滑，并且代码采用极小的步长，则您可以尝试用非专用于刚性问题的求解器来解算刚性问题。改用刚性求解器 <code>ode15s</code>、<code>ode23s</code>、<code>ode23t</code> 或 <code>ode23tb</code> 中的一个。</p>
如果我有最终值但没有初始值，应该怎么办？	ODE 套件的所有求解器都允许您按时间向前或向后解算。这些求解器的语法为 <code>[t,y] = ode45(odefun, [t0 tf],y0);</code> ，此语法接受 <code>t0 > tf</code> 。

疑问或问题	回答
我的积分过程异常缓慢，使用了太多的时间步。	<p>首先，检查并确保 <code>tspan</code> 不是太长。请记住，求解器使用多个必要的时点来生成平滑的解。如果 ODE 函数在相比 <code>tspan</code> 非常短的时间标量上发生变化，则求解器会使用大量时间步长。长时间的积分是一个非常复杂的问题。请将 <code>tspan</code> 划分多个较小的片段。</p> <p>如果 ODE 函数在 <code>tspan</code> 区间中没有发生明显变化，则可能是因为问题是刚性问题所致。尝试使用刚性求解器 <code>ode15s</code>、<code>ode23s</code>、<code>ode23t</code> 或 <code>ode23tb</code> 中的一个。</p> <p>最后，确保以有效方式编写 ODE 函数。求解器会在 ODE 函数中多次计算导数。数值积分的成本在很大程度上依赖于计算 ODE 函数的成本。请不要在每次计算中重新计算复杂的常量参数，而应将其存储到全局变量中，或者对这些参数仅计算一次并将其传递给嵌套函数。</p>
我确定解在 t 时刻 ($t_0 \leq t \leq t_f$) 发生了急剧变化，但积分器步已完成，并且没有“注意到”这一点。	<p>如果您确定在 t 时刻出现急剧变化，请尝试将 <code>tspan</code> 区间分解为两个片段 $[t_0 t]$ 和 $[t t_f]$，并调用两次积分器或使用 <code>odextend</code> 继续积分。</p> <p>如果微分方程具有周期性系数或解，通过将最大步长限制为此周期的长度，确保求解器不会跨越多个周期。</p>

问题类型

求解器可否处理使用直线法离散的偏微分方程 (PDE)?	<p>可以，因为离散生成 ODE 方程组。根据离散的情况，可能具有一种 ODE 求解器已考虑到的与质量矩阵相关的方程组形式。此方程组往往为刚性方程组。如果 PDE 为抛物线，或现象发生的时间跨度极其不同（如流体中的化学反应），即属于这种情况。在这种情况下，请使用下列四个刚性求解器中的一个：<code>ode15s</code>、<code>ode23s</code>、<code>ode23t</code> 或 <code>ode23tb</code>。</p> <p>如果存在多个方程，则使用 <code>JPattern</code> 选项来指定 Jacobian 稀疏模式。这对解算成败可能起到决定性的作用，因为它能防止过高的计算开销。键入 <code>edit burgersode</code>，查看使用 <code>JPattern</code> 的示例。</p> <p>如果方程组不是刚性方程组，或者刚性不太强，则 <code>ode23</code> 和 <code>ode45</code> 比刚性求解器 <code>ode15s</code>、<code>ode23s</code>、<code>ode23t</code> 和 <code>ode23tb</code> 更加有效。</p> <p>您可以使用 MATLAB PDE 求解器 <code>pdepe</code> 直接解算一维抛物线-椭圆偏微分方程。</p>
-----------------------------	---

可否求一组示例数据的积分？

不能直接求积分。相反，应通过插值或其他某种方案将数据表示为函数以便拟合数据。此函数的平滑性至关重要。样条曲线等分段多项式拟合看似平滑，但对求解器而言却并非如此；当拟合导数包含曲折时，求解器采用较小的步长。请使用平滑的函数来表示数据，或者使用对平滑性不太敏感的某个低阶求解器 (**ode23**、**ode23s**、**ode23t**、**ode23tb**)。请参阅“带有时变项的 ODE”中的示例。

另请参阅

[deval](#) | [odeget](#) | [odeset](#) | [odextend](#)

详细信息

- “选择 ODE 求解器” (第 11-2 页)
- “ODE 选项摘要” (第 11-9 页)
- “ODE 事件位置” (第 11-11 页)
- “非负 ODE 解” (第 11-32 页)

微分方程

此示例说明如何使用 MATLAB® 构造几种不同类型的微分方程并求解。MATLAB 提供了多种数值算法来求解各种微分方程：

- 初始值问题
- 边界值问题
- 时滞微分方程
- 偏微分方程

初始值问题

vanderpoldemo 是用于定义 van der Pol 方程的函数

$$\frac{d^2y}{dt^2} - \mu(1 - y^2)\frac{dy}{dt} + y = 0.$$

```
type vanderpoldemo
```

```
function dydt = vanderpoldemo(t,y,Mu)
%VANDERPOLDEMO Defines the van der Pol equation for ODEDEMO.

% Copyright 1984-2014 The MathWorks, Inc.

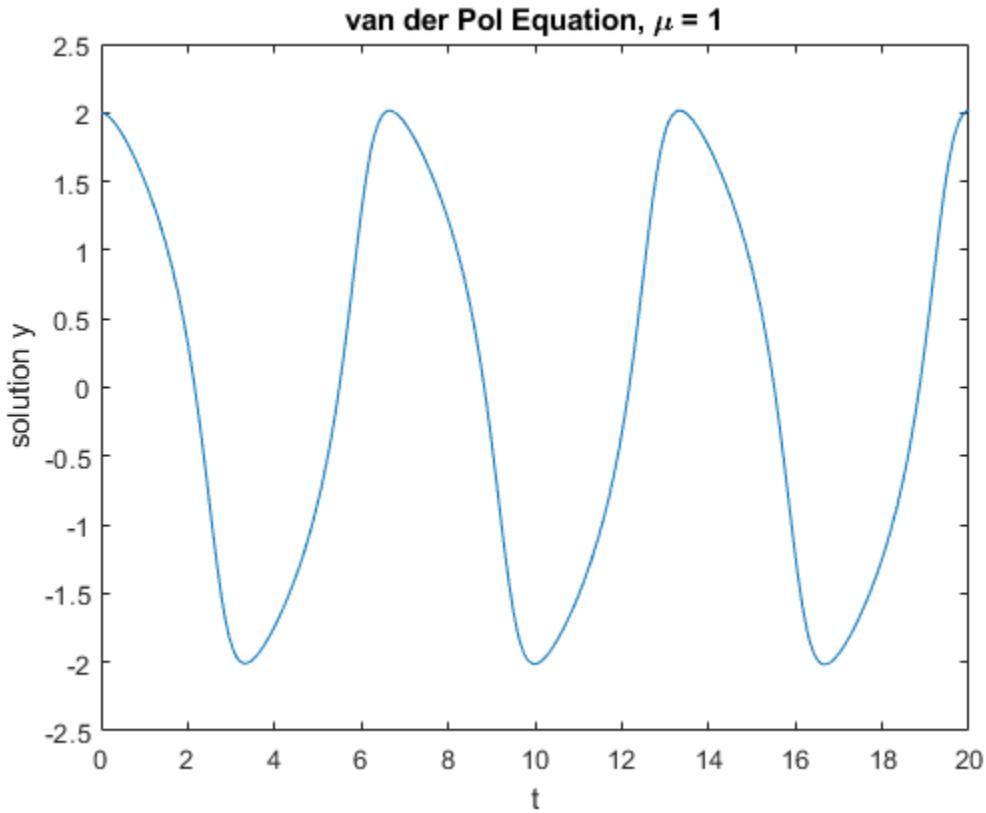
dydt = [y(2); Mu*(1-y(1)^2)*y(2)-y(1)];
```

该方程写作包含两个一阶常微分方程 (ODE) 的方程组。将针对参数 μ 的不同值计算这些方程。为了实现更快的积分，您应该根据 μ 的值选择合适的求解器。

当 $\mu = 1$ 时，任何 MATLAB ODE 求解器都能有效地求解 van der Pol 方程。**ode45** 求解器就是其中之一。该方程在域 $[0, 20]$ 中求解，初始条件为 $y(0) = 2$ 和 $\frac{dy}{dt} \Big|_{t=0} = 0$ 。

```
tspan = [0 20];
y0 = [2; 0];
Mu = 1;
ode = @(t,y) vanderpoldemo(t,y,Mu);
[t,y] = ode45(ode, tspan, y0);

% Plot solution
plot(t,y(:,1))
xlabel('t')
ylabel('solution y')
title('van der Pol Equation, \mu = 1')
```

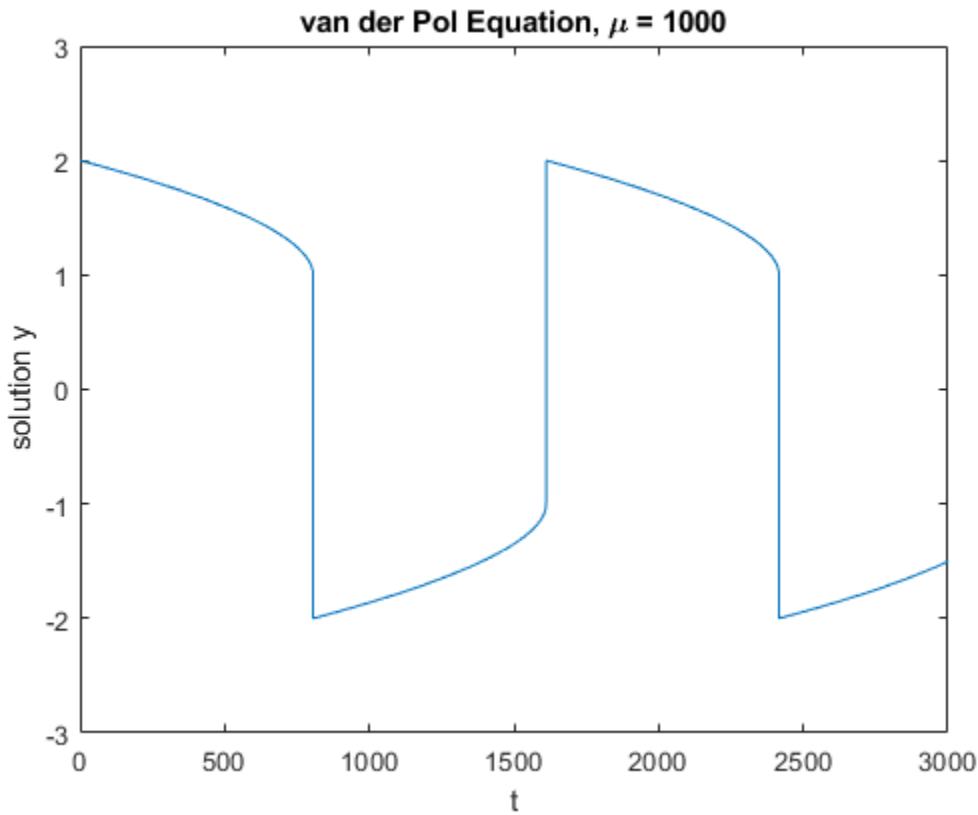


对于较大的 μ , 问题将变为刚性。此标签表示拒绝使用普通方法计算的问题。这种情况下, 要实现快速积分, 需要使用特殊的数值方法。**ode15s**、**ode23s**、**ode23t** 和 **ode23tb** 函数可有效地求解刚性问题。

当 $\mu = 1000$ 时, van der Pol 方程的求解使用 **ode15s**, 初始条件相同。您需要将时间范围大幅度延长到 $[0, 3000]$ 才能看到解的周期性变化。

```
tspan = [0, 3000];
y0 = [2; 0];
Mu = 1000;
ode = @(t,y) vanderpoldemo(t,y,Mu);
[t,y] = ode15s(ode, tspan, y0);

plot(t,y(:,1))
title('van der Pol Equation, \mu = 1000')
axis([0 3000 -3 3])
xlabel('t')
ylabel('solution y')
```



边界值问题

bvp4c 和 **bvp5c** 可以求解常微分方程的边界值问题。

示例函数 **twoode** 将一个微分方程写作包含两个一阶 ODE 的方程组。此微分方程为

$$\frac{d^2y}{dt^2} + |y| = 0.$$

type twoode

```
function dydx = twoode(x,y)
%TWOODE Evaluate the differential equations for TWOBVP.
%
% See also TWOBC, TWOBVP.

% Lawrence F. Shampine and Jacek Kierzenka
% Copyright 1984-2014 The MathWorks, Inc.
```

dydx = [y(2); -abs(y(1))];

函数 **twobc** 求解该问题的边界条件为: $y(0) = 0$ 和 $y(4) = -2$ 。

type twobc

```
function res = twobc(ya,yb)
%TWOBC Evaluate the residual in the boundary conditions for TWOBVP.
```

```
%  
% See also TWOODE, TWOBVP.  
  
% Lawrence F. Shampine and Jacek Kierzenka  
% Copyright 1984-2014 The MathWorks, Inc.
```

```
res = [ ya(1); yb(1) + 2 ];
```

在调用 **bvp4c** 之前，您必须为要在网格中表示的解提供一个猜想值。然后，求解器就像对解进行平滑处理一样修改网格。

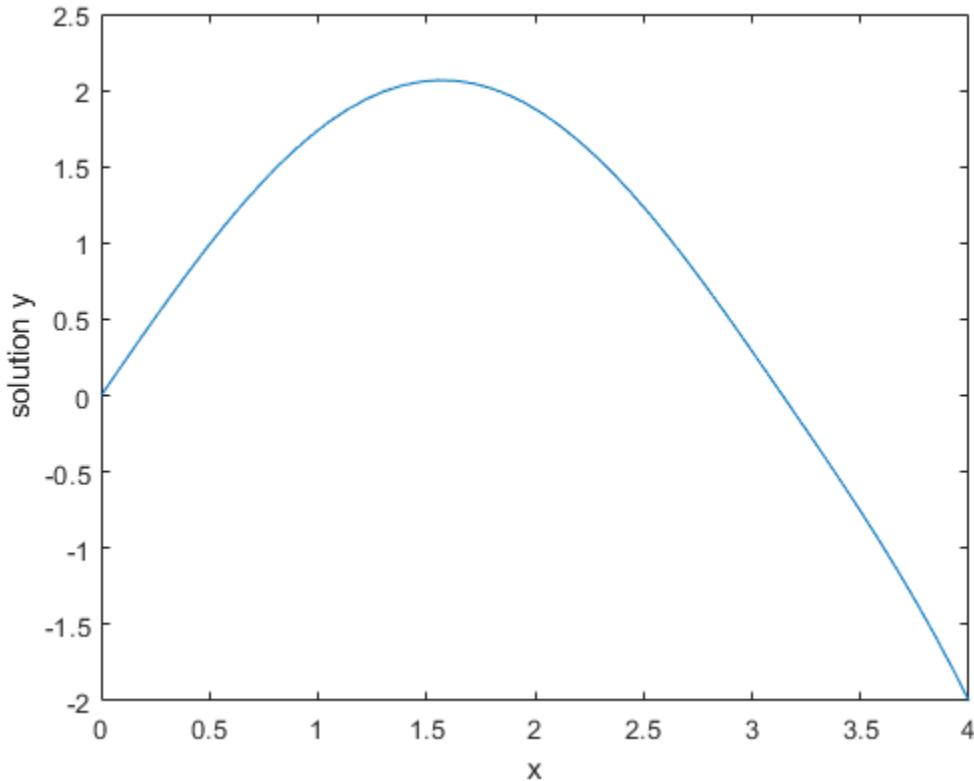
bvpinit 函数以您可以传递给求解器 **bvp4c** 的形式设定初始猜想值。对于 **[0 1 2 3 4]** 的网格以及 $y(x) = 1$ 和 $y'(x) = 0$ 的常量猜想值，对 **bvpinit** 的调用为：

```
solinit = bvpinit([0 1 2 3 4],[1; 0]);
```

利用这个初始猜想值，您可以使用 **bvp4c** 对该问题求解。使用 **deval** 计算 **bvp4c** 在某些点返回的解，然后绘制结果值。

```
sol = bvp4c(@twoode, @twobc, solinit);
```

```
xint = linspace(0, 4, 50);  
yint = deval(sol, xint);  
plot(xint, yint(1,:));  
xlabel('x')  
ylabel('solution y')  
hold on
```



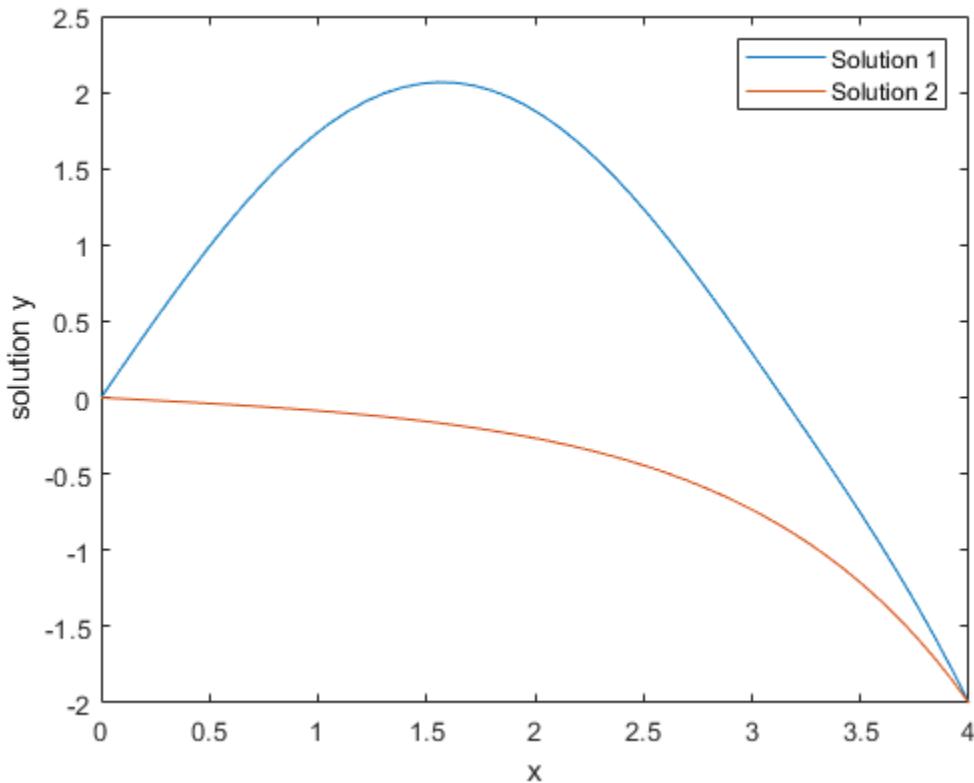
此特定的边界值问题实际上有两种解。通过将初始猜想值更改为 $y(x) = -1$ 和 $y'(x) = 0$ ，可以求出另一个解。

```

solinit = bvpinit([0 1 2 3 4],[-1; 0]);
sol = bvp4c(@twoode,@twobc,solinit);

xint = linspace(0,4,50);
yint = deval(sol,xint);
plot(xint,yint(1,:));
legend('Solution 1','Solution 2')
hold off

```



时滞微分方程

dde23、**ddesd** 和 **ddensd** 可以求解具有各种时滞的时滞微分方程。示例 **ddex1**、**ddex2**、**ddex3**、**ddex4** 和 **ddex5** 构成了这些求解器的迷你使用教程。

ddex1 示例说明如何求解微分方程组

$$\begin{aligned}
 y_1'(t) &= y_1(t-1) \\
 y_2'(t) &= y_1(t-1) + y_2(t-0.2) \\
 y_3'(t) &= y_2(t).
 \end{aligned}$$

您可以使用匿名函数表示这些方程

```
ddex1fun = @(t,y,Z) [Z(1,1); Z(1,1)+Z(2,2); y(2)];
```

问题的历史解 ($t \leq 0$ 时) 固定不变:

$$y_1(t) = 1$$

$$y_2(t) = 1$$

$$y_3(t) = 1.$$

您可以将历史解表示为由 1 组成的向量。

```
ddex1hist = ones(3,1);
```

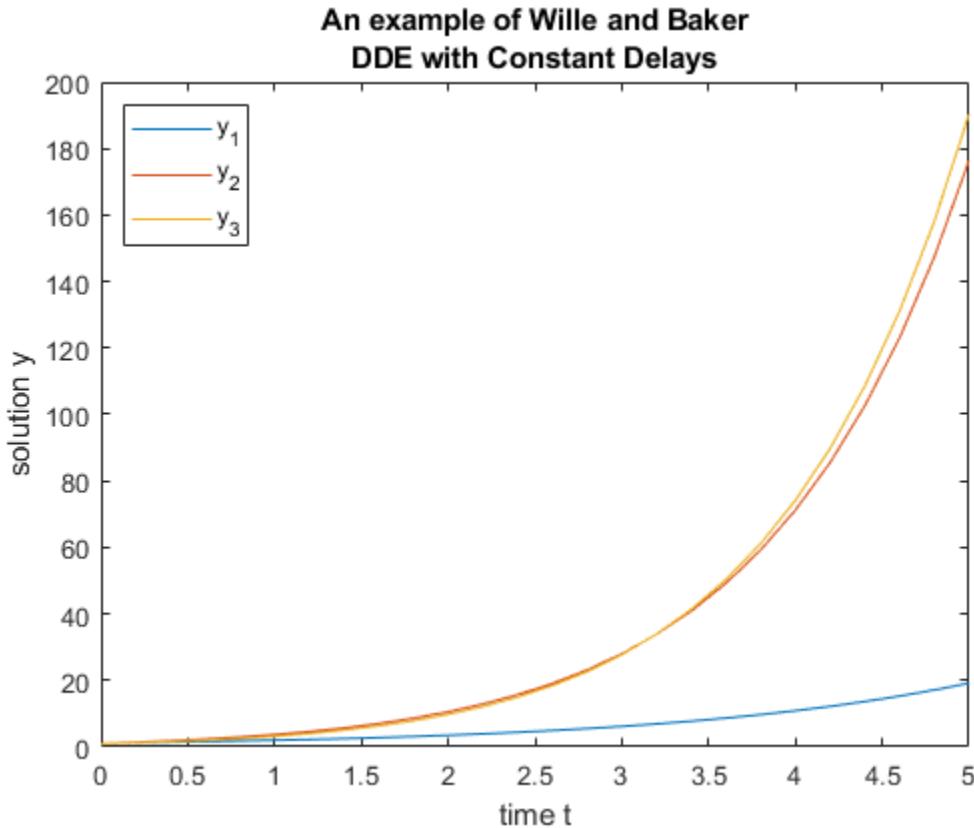
采用二元素向量表示方程组中的时滞。

```
lags = [1 0.2];
```

将函数、时滞、历史解和积分区间 $[0, 5]$ 作为输入传递给求解器。求解器在整个积分区间生成适合绘图的连续解。

```
sol = dde23(ddex1fun, lags, ddex1hist, [0 5]);
```

```
plot(sol.x,sol.y);
title({'An example of Wille and Baker', 'DDE with Constant Delays'});
xlabel('time t');
ylabel('solution y');
legend('y_1','y_2','y_3','Location','NorthWest');
```



偏微分方程

pdepe 使用一个空间变量和时间对偏微分方程求解。示例 **pdex1**、**pdex2**、**pdex3**、**pdex4** 和 **pdex5** 构成了 **pdepe** 的迷你使用教程。

此示例问题使用函数 **pdex1pde**、**pdex1ic** 和 **pdex1bc**。

pdex1pde 定义微分方程

$$\pi^2 \frac{\partial u}{\partial t} = \frac{\partial}{\partial x} \left(\frac{\partial u}{\partial x} \right).$$

type pdex1pde

```
function [c,f,s] = pdex1pde(x,t,u,DuDx)
%PDEX1PDE Evaluate the differential equations components for the PDEX1 problem.
%
% See also PDEPE, PDEX1.

% Lawrence F. Shampine and Jacek Kierzenka
% Copyright 1984-2014 The MathWorks, Inc.

c = pi^2;
f = DuDx;
s = 0;
```

pdex1ic 设置初始条件

$$u(x, 0) = \sin \pi x.$$

type pdex1ic

```
function u0 = pdex1ic(x)
%PDEX1IC Evaluate the initial conditions for the problem coded in PDEX1.
%
% See also PDEPE, PDEX1.

% Lawrence F. Shampine and Jacek Kierzenka
% Copyright 1984-2014 The MathWorks, Inc.
```

`u0 = sin(pi*x);`

pdex1bc 设置边界条件

$$u(0, t) = 0,$$

$$\pi e^{-t} + \frac{\partial}{\partial x} u(1, t) = 0.$$

type pdex1bc

```
function [pl,ql,pr,qr] = pdex1bc(xl,ul,xr,ur,t)
%PDEX1BC Evaluate the boundary conditions for the problem coded in PDEX1.
%
% See also PDEPE, PDEX1.

% Lawrence F. Shampine and Jacek Kierzenka
% Copyright 1984-2014 The MathWorks, Inc.
```

```

pl = ul;
ql = 0;
pr = pi * exp(-t);
qr = 1;

```

pdepe 需要提供空间离散 x 和时间向量 t (您要获取解快照的时间点)。使用包含 20 个节点的网格求解此问题，并请求五个 t 值的解。提取解的第一个分量并绘图。

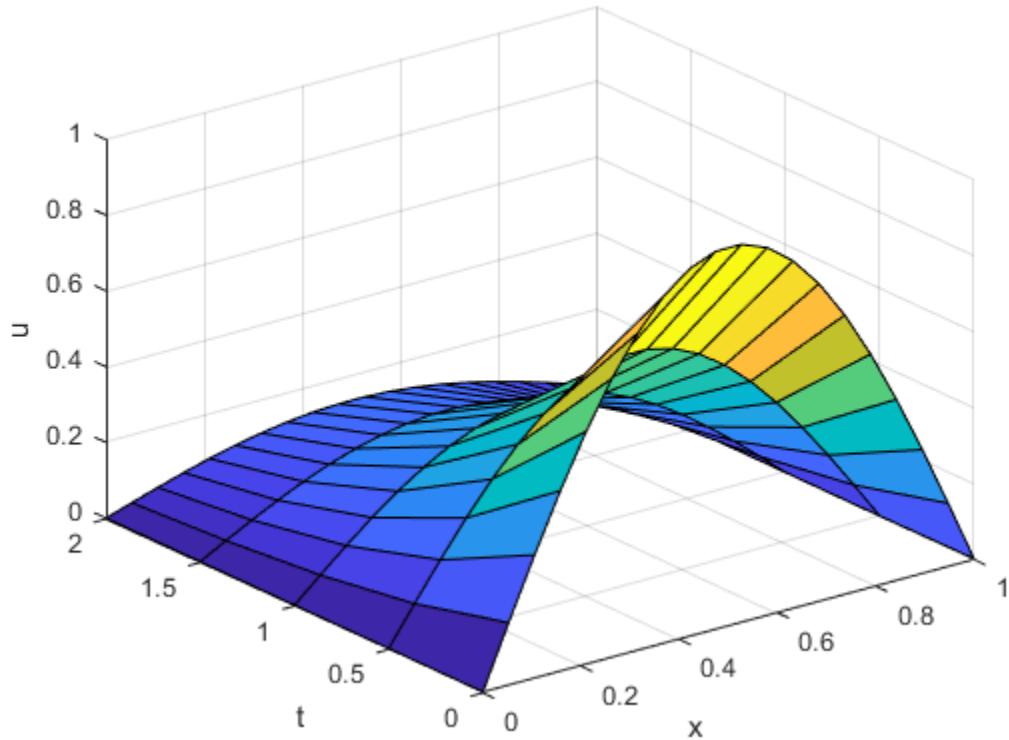
```

x = linspace(0,1,20);
t = [0 0.5 1 1.5 2];
sol = pdepe(0,@pdex1pde,@pdex1ic,@pdex1bc,x,t);

u1 = sol(:,:,1);

surf(x,t,u1);
xlabel('x');
ylabel('t');
zlabel('u');

```



另请参阅

[bvp4c](#) | [ode45](#) | [pdepe](#)

详细信息

- “[选择 ODE 求解器](#)” (第 11-2 页)

- “求解边界值问题” (第 12-2 页)
- “求解偏微分方程” (第 13-2 页)

求解捕食者-猎物方程

此示例说明如何使用 `ode23` 和 `ode45` 求解表示捕食者/猎物模型的微分方程。这两个函数用于对使用可变步长大小 Runge-Kutta 积分方法的常微分方程求数值解。`ode23` 使用一对简单的 2 阶和 3 阶公式实现中等精度，`ode45` 使用一对 4 阶和 5 阶公式实现更高的精度。

以名为 **Lotka-Volterra 方程**，也即**捕食者-猎物模型**的一对一阶常微分方程为例：

$$\begin{aligned}\frac{dx}{dt} &= x - \alpha xy \\ \frac{dy}{dt} &= -y + \beta xy.\end{aligned}$$

变量 x 和 y 分别计算猎物和捕食者的数量。二次交叉项表示物种之间的交叉。当没有捕食者时，猎物数量将增加，当猎物匮乏时，捕食者数量将减少。

编写方程代码

为了模拟系统，需要创建一个函数，以返回给定状态和时间值时的状态导数的列向量。在 MATLAB 中，两个变量 x 和 y 可以表示为向量 y 中的前两个值。同样，导数是向量 yp 中的前两个值。函数必须接受 t 和 y 的值，并在 yp 中返回公式生成的值。

```
yp(1) = (1 - alpha*y(2))*y(1)
yp(2) = (-1 + beta*y(1))*y(2)
```

在此示例中，公式包含在名为 `lotka.m` 的文件中。此文件使用 $\alpha = 0.01$ 和 $\beta = 0.02$ 的参数值。

```
type lotka
```

模拟系统

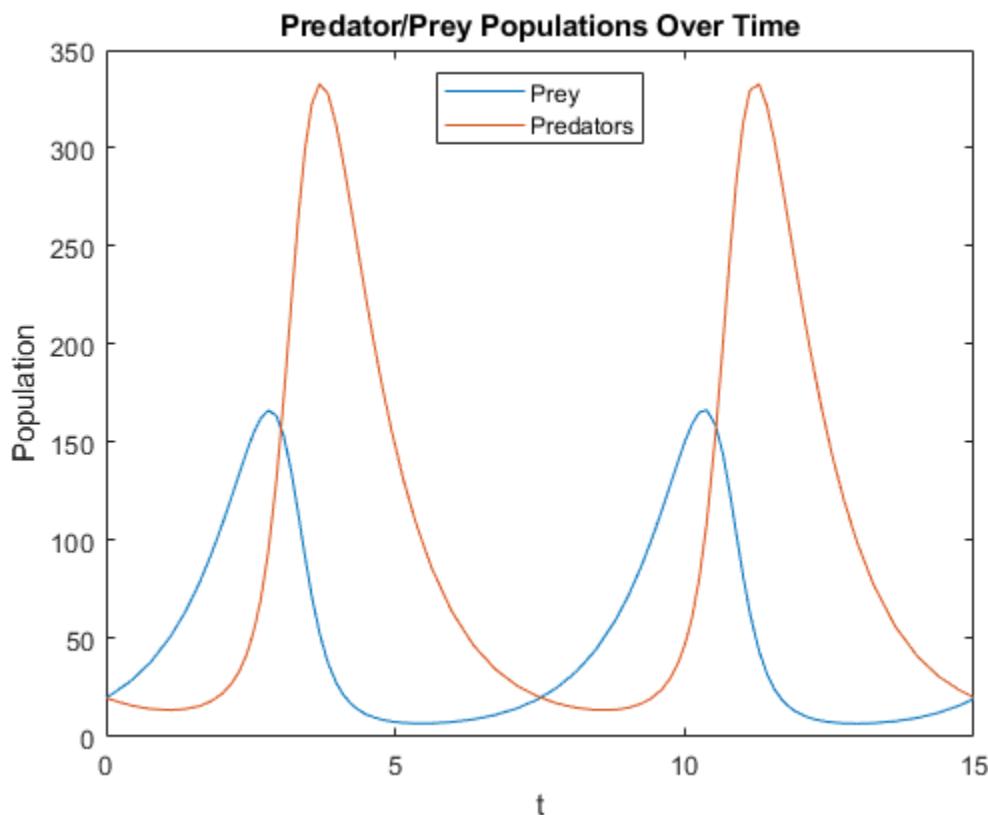
使用 `ode23` 在区间 $0 < t < 15$ 中求解 `lotka` 中定义的微分方程。使用初始条件 $x(0) = y(0) = 20$ ，使捕食者和猎物的数量相等。

```
t0 = 0;
tfinal = 15;
y0 = [20; 20];
[t,y] = ode23(@lotka,[t0 tfinal],y0);
```

绘制结果

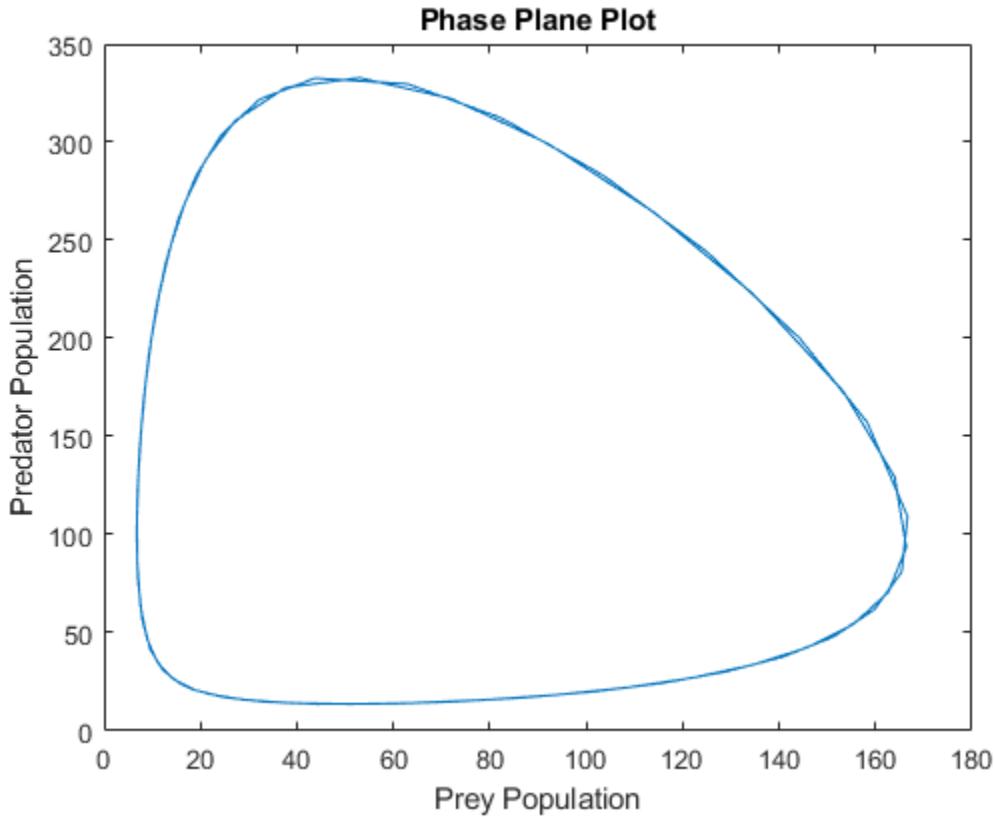
绘制两个种群数量对时间的图。

```
plot(t,y)
title('Predator/Prey Populations Over Time')
xlabel('t')
ylabel('Population')
legend('Prey','Predators','Location','North')
```



现在绘制两个种群数量的相对关系图。生成的相平面图非常清晰地表明了二者数量之间的循环关系。

```
plot(y(:,1),y(:,2))
title('Phase Plane Plot')
xlabel('Prey Population')
ylabel('Predator Population')
```

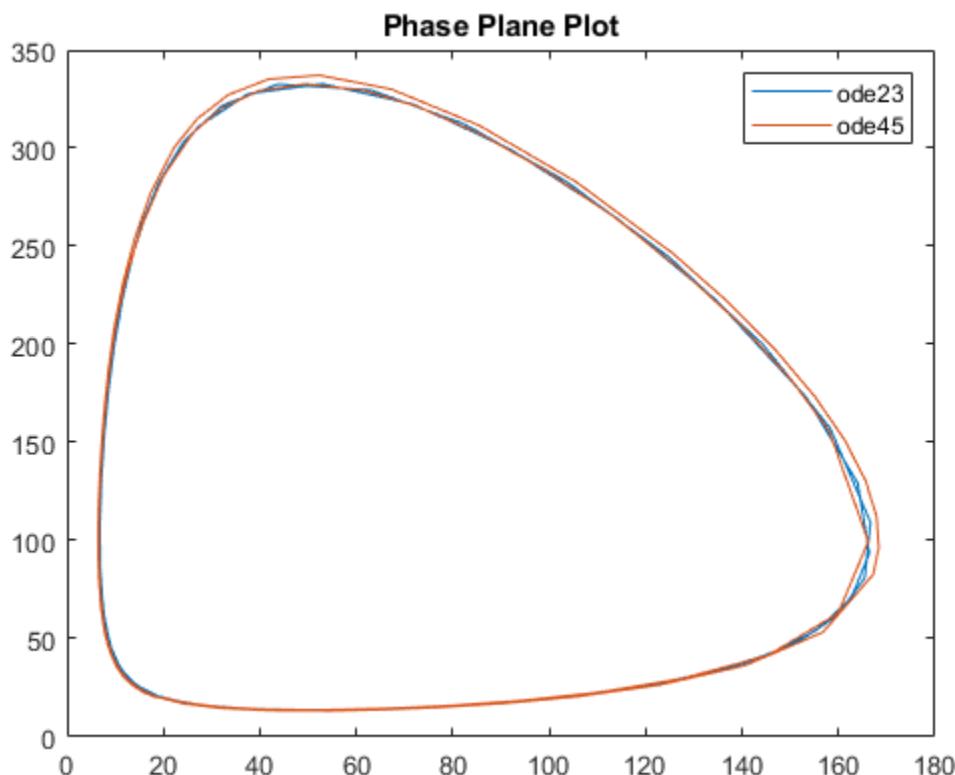


比较不同求解器的结果

使用 `ode45` 而不是 `ode23` 再次求解该方程组。`ode45` 求解器的每一步都需要更长的时间，但它的步长也更大。然而，`ode45` 的输出是平滑的，因为默认情况下，此求解器使用连续展开公式在每个步长范围内的四个等间距时间点生成输出。（您可以使用 '`Refine`' 选项调整时间点数。）绘制两个解进行比较。

```
[T,Y] = ode45(@lotka,[t0 tfinal],y0);
```

```
plot(y(:,1),y(:,2),'-',Y(:,1),Y(:,2),'-');
title('Phase Plane Plot')
legend('ode23','ode45')
```



结果表明，使用不同的数值方法求解微分方程会产生略微不同的答案。

另请参阅

[ode23](#) | [ode45](#)

详细信息

- “选择 ODE 求解器” (第 11-2 页)
- “求解非刚性 ODE” (第 11-17 页)

边界值问题 (BVP)

- “求解边界值问题” (第 12-2 页)
- “对具有两个解的 BVP 求解” (第 12-7 页)
- “求解具有未知参数的 BVP” (第 12-10 页)
- “使用延拓求解 BVP 问题” (第 12-14 页)
- “使用延拓验证 BVP 一致性” (第 12-18 页)
- “求解具有奇异项的 BVP” (第 12-23 页)
- “求解具有多边界条件的 BVP” (第 12-27 页)

求解边界值问题

在边界值问题 (BVP) 中，目标是求常微分方程 (ODE) 的解，该解还需满足某些指定的边界条件。边界条件指定积分区间中两个或多个位置处的解的值之间的关系。在最简单的情形中，边界条件适用于区间的开始和结束（即边界）。

MATLAB BVP 求解器 **bvp4c** 和 **bvp5c** 用于处理以下形式的 ODE 方程组：

$$y' = f(x, y)$$

其中：

- x 是自变量
- y 是因变量
- y' 表示 y 关于 x 的导数，也写为 dy/dx

边界条件

在两点 BVP 的最简单情形中，ODE 的解在区间 $[a, b]$ 中求得，并且必须满足边界条件

$$g(y(a), y(b)) = 0 .$$

要指定给定 BVP 的边界条件，您必须：

- 编写 `res = bcfun(ya,yb)` 形式的函数，如果涉及未知参数，则使用 `res = bcfun(ya,yb,p)` 形式。将此函数作为第二个输入参数提供给求解器。该函数返回 `res`，这是在边界点处的解的残差值。例如，如果 $y(a) = 1$ 且 $y(b) = 0$ ，则边界条件函数为

```
function res = bcfun(ya,yb)
res = [ya(1)-1
       yb(1)];
end
```

- 在解的初始估计值中，网格中的第一个和最后一个点指定强制执行边界条件的点。对于上述边界条件，您可以指定 `bvpinit(linspace(a,b,5),yinit)` 来对 a 和 b 强制执行边界条件。

MATLAB 中的 BVP 求解器还适用于包含下列各项的其他类型问题：

- 未知参数 p
- 解具有奇异性
- 多点条件（将积分区间分成若干区域的内边界）

在多点边界条件的情形下，边界条件应用于积分区间中两个以上的点。例如，可能要求在区间的开始处、中间处和结束处的解为零。有关如何指定多点边界条件的详细信息，请参阅 `bvpinit`。

解的初始估计值

与初始值问题不同，边界值问题的解可以是：

- 无解
- 有限个解
- 无限多个解

求解 BVP 过程的重要部分是提供所需解的估计值。估计值的准确与否对求解器性能甚至是能否成功计算来说至关重要。

使用 **bvpinit** 函数为解的初始估计值创建结构体。求解器 **bvp4c** 和 **bvp5c** 接受此结构体作为第三个输入参数。

为解创建良好的初始估计值更像是一门艺术，而不是一门科学。不过，仍有一些常规指导原则，包括：

- 确保初始估计值满足边界条件，因为解也需要满足这些边界条件。如果问题包含未知参数，则参数的初始估计值也应该满足边界条件。
- 尝试将关于实际问题或预期解的信息尽可能多地纳入初始估计值。例如，如果解应该振荡或者有一定数量的符号变化，则初始估计值也应该如此。
- 考虑网格点的位置（解的初始估计值的 x 坐标）。BVP 求解器会在求解过程中调整这些点，因此您不需要指定太多网格点。最佳做法是在解会快速变化的位置附近指定几个网格点。
- 如果在较小区间内有一个已知的、更简单的解，则将它用作较大区间内的初始估计值。通常，您可以将一个问题作为一系列相对简单的问题来求解，这种做法称为延拓。使用延拓时，可以使用一个问题的解作为求解下一个问题的初始估计值，从而将一系列简单问题连接起来。

查找未知参数

通常，BVP 会涉及需要同时求解的未知 p 参数。ODE 和边界条件变为

$$\begin{aligned}y' &= f(x, y, p) \\g(y(a), y(b), p) &= 0\end{aligned}$$

在此情况下，边界条件必须足以确定参数 p 的值。

您必须为求解器提供任何未知参数的初始估计值。当调用 **bvpinit** 来创建结构体 **solinit** 时，请在第三个输入参数 **parameters** 中指定初始估计值向量。

```
solinit = bvpinit(x,v,parameters)
```

此外，用于编写 ODE 方程和边界条件代码的函数 **odefun** 和 **bcfun** 都必须具有第三个参数。

```
dydx = odefun(x,y,parameters)  
res = bcfun(ya,yb,parameters)
```

求解微分方程时，求解器会调整未知参数的值以满足边界条件。求解器会在 **sol.parameters** 中返回未知参数的最终值。

奇异 BVP

bvp4c 和 **bvp5c** 可对以下形式的一类奇异 BVP 求解

$$\begin{aligned}y' &= \frac{1}{x}Sy + f(x, y), \\0 &= g(y(0), y(b)).\end{aligned}$$

该求解器还可以接受以下形式的问题的未知参数

$$\begin{aligned}y' &= \frac{1}{x}Sy + f(x, y, p), \\0 &= g(y(0), y(b), p).\end{aligned}$$

奇异问题必须位于 $[0,b]$ 区间上，且 $b > 0$ 。使用 **bvpset** 将常量矩阵 S 作为 'SingularTerm' 选项的值传递给求解器。 $x = 0$ 时的边界条件必须与平滑解 $Sy(0) = 0$ 的必要条件一致。解的初始估计值也应该满足此条件。

当您求解奇异 BVP 时，求解器要求您的函数 **odefun(x,y)** 只返回方程中 $f(x, y)$ 项的值。涉及 S 的项由求解器使用 'SingularTerm' 选项单独处理。

BVP 求解器的选择

MATLAB 提供了求解器 **bvp4c** 和 **bvp5c** 来求解 BVP。在大多数情况下，您可以互换使用这些求解器。这些求解器之间的主要区别在于 **bvp4c** 实现四阶公式，而 **bvp5c** 实现五阶公式。

除两个求解器之间的误差容限含义以外，**bvp5c** 函数的使用方法与 **bvp4c** 完全类似。如果 $S(x)$ 近似于解 $y(x)$ ，则 **bvp4c** 控制残差 $|S'(x) - f(x, S(x))|$ 。这种方法间接控制真误差 $|y(x) - S(x)|$ 。使用 **bvp5c** 直接控制真误差。

求解器	说明
bvp4c	<p>bvp4c 是一个有限差分代码，此代码实现 3 阶段 Lobatto IIIa 公式。这是配置公式，并且配置多项式会提供在整个积分区间中处于四阶精度的 C^1 连续解。网格选择和误差控制均基于连续解的残差。</p> <p>配置方法使用点网格将积分区间分为子区间。通过对源于边界条件以及所有子区间上配置条件的线性代数方程全局组求解，求解器会确定数值解。然后，求解器会估计每个子区间上数值解的误差。如果解不满足容差标准，则求解器会调整网格并重复计算过程。您必须提供初始网格的点以及网格点处解的初始近似估计。</p>
bvp5c	<p>bvp5c 是一个有限差分代码，此代码实现 4 阶段 Lobatto IIIa 公式。这是配置公式，并且配置多项式会提供在整个 $[a,b]$ 中具有一致五阶精度的 C^1 连续解。该公式作为隐式 Runge-Kutta 公式实现。</p> <p>bvp5c 直接对代数方程求解；然而，bvp4c 使用解析压缩法。bvp4c 直接处理未知参数；而 bvp5c 使用未知参数的平凡微分方程扩充方程组。</p>

解的计算

在 **bvp4c** 和 **bvp5c** 中实现的配置方法在积分区间 $[a,b]$ 上产生 C^1 连续解。可以使用求解器返回的辅助函数 **deval** 和结构体 **sol**，在 $[a,b]$ 中的任意点计算 $S(x)$ 近似解。例如，要在网格点 **xint** 上计算解 **sol**，请使用以下命令

```
Sxint = deval(sol,xint)
```

deval 函数已向量化。对于向量 **xint**，**Sxint** 的第 **i** 列与解 $y(xint(i))$ 近似。

BVP 示例和文件

我们可以通过几个示例文件来很好地了解如何求解最常见的 BVP 问题。要方便地查看和运行示例，可以使用 **Differential Equations Examples** App。要运行此 App，请键入

odeexamples

要打开单独的示例文件进行编辑, 请键入

edit exampleFileName.m

要运行示例, 请键入

exampleFileName

下表包含可用的 BVP 示例文件及其使用的求解器和选项的列表。

示例文件	使用的求解器	指定的选项	说明	示例链接
emdenbvp	bvp4c 或 bvp5c	• 'SingularTerm'	埃姆登方程, 奇异 BVP	"求解具有奇异项的 BVP" (第 12-23 页)
fsbvp	bvp4c 或 bvp5c	—	无限区间上的 Falkner-Skan BVP	"使用延拓验证 BVP 一致性" (第 12-18 页)
mat4bvp	bvp4c 或 bvp5c	—	马蒂厄方程的第四个特征函数	"求解具有未知参数的 BVP" (第 12-10 页)
rcbvp	bvp4c 和 bvp5c	• 'FJacobian' • 'AbsTol' • 'RelTol' • 'Stats'	比较 bvp4c 和 bvp5c 控制的误差的示例	"比较 bvp4c 和 bvp5c 求解器" (bvp4c) "比较 bvp4c 和 bvp5c 求解器" (bvp5c)
shockbpvp	bvp4c 或 bvp5c	• 'FJacobian' • 'BCJacobian' • 'Vectorized'	冲击层位于 $x = 0$ 的解	"使用延拓求解 BVP 问题" (第 12-14 页)
twobvp	bvp4c	—	只有两个解的 BVP	"对具有两个解的 BVP 求解" (第 12-7 页)
threebpvp	bvp4c 或 bvp5c	—	三点边界值问题	"求解具有多边界条件的 BVP" (第 12-27 页)

参考

- [1] Ascher, U., R. Mattheij, and R. Russell. "Numerical Solution of Boundary Value Problems for Ordinary Differential Equations." Philadelphia, PA: SIAM, 1995, p. 372.
- [2] Shampine, L.F., and J. Kierzenka. "A BVP Solver based on residual control and the MATLAB PSE." ACM Trans. Math. Softw. Vol. 27, Number 3, 2001, pp. 299–316.
- [3] Shampine, L.F., M.W. Reichelt, and J. Kierzenka. "Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB with bvp4c." *MATLAB File Exchange*, 2004.

[4] Shampine, L.F., and J. Kierzenka. "A BVP Solver that Controls Residual and Error." *J. Numer. Anal. Ind. Appl. Math.* Vol. 3(1-2), 2008, pp. 27–41.

另请参阅

bvp4c | bvp5c | bvpinit | bvpset | ode45 | pdepe

对具有两个解的 BVP 求解

此示例使用 **bvp4c** 和两个不同的初始估计值来求 BVP 问题的两个解。

假设有以下微分方程：

$$y'' + e^y = 0.$$

此方程具有如下边界条件：

$$y(0) = y(1) = 0.$$

要在 MATLAB 中对该方程求解，您需要先编写方程和边界条件的代码，然后为解生成合适的初始估计值，再调用边界值问题求解器 **bvp4c**。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写方程代码

创建一个函数以编写方程代码。此函数应具有签名 **dydx = bvpfun(x,y)** 或 **dydx = bvpfun(x,y,parameters)**，其中：

- **x** 是自变量。
- **y** 是解（因变量）。
- **parameters** 是未知参数值的向量（可选）。

求解器会自动将这些输入传递给该函数，但是变量名称决定如何编写方程代码。在本例中，可以将二阶方程重写为一阶方程组

$$y_1' = y_2,$$

$$y_2' = -e^{y_1}.$$

用于编写这些方程代码的函数为

```
function dydx = bvpfun(x,y)
dydx = [y(2)
        -exp(y(1))];
end
```

编写边界条件代码

对于像此问题中的两点边界值条件，边界条件函数应该具有签名 **res = bcfun(ya,yb)** 或 **res = bcfun(ya,yb,parameters)**，具体取决于是否涉及未知参数。**ya** 和 **yb** 是求解器自动传递给函数的列向量，**bcfun** 返回边界条件中的残差。

对于边界条件 $y(0) = y(1) = 0$ ，**bcfun** 函数指定两个边界上的残差值都为零。在您的初始估计值中，这些残差值会强制应用于您指定给 **bvpinit** 的第一个和最后一个网格点。此问题的初始网格应该有 **x(1) = 0** 和 **x(end) = 1**。

```
function res = bcfun(ya,yb)
res = [ya(1)
       yb(1)];
end
```

获取初始估计值

调用 **bvpinit** 以生成解的初始估计值。**x** 的网格不需要有很多点，但是第一个点必须为 0，而最后一个点必须为 1，以正确指定边界条件。对 **y** 使用初始估计值，其中第一个分量为稍大于零的正数，第二个分量为零。

```
xmesh = linspace(0,1,5);
solinit = bvpinit(xmesh, [0.1 0]);
```

求解方程

使用 **bvp4c** 求解器求解 BVP。

```
sol1 = bvp4c(@bvpfun, @bcfun, solinit);
```

使用不同的初始估计值

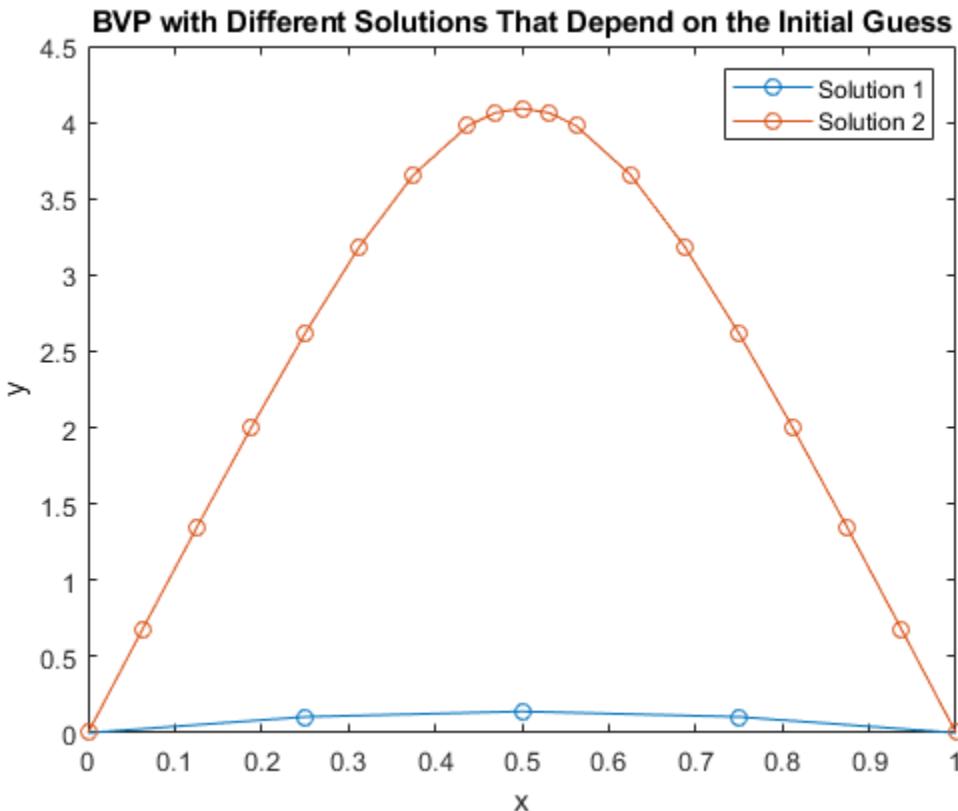
使用解的不同初始估计值第二次求解 BVP。

```
solinit = bvpinit(xmesh, [3 0]);
sol2 = bvp4c(@bvpfun, @bcfun, solinit);
```

对解进行比较

绘制 **bvp4c** 针对不同的初始条件所计算的解。这两个解都满足规定的边界条件，但它们之间有不同行为。由于解并不始终唯一，不同行为展现出为解提供良好初始估计值的重要性。

```
plot(sol1.x,sol1.y(1,:),'-o',sol2.x,sol2.y(1,:),'-o')
title('BVP with Different Solutions That Depend on the Initial Guess')
xlabel('x')
ylabel('y')
legend('Solution 1','Solution 2')
```



局部函数

此处列出了 BVP 求解器 `bvp4c` 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```
function dydx = bvpfun(x,y) % equation being solved
dydx = [y(2)
        -exp(y(1))];
end
%
function res = bcfun(ya,yb) % boundary conditions
res = [ya(1)
        yb(1)];
end
%
```

另请参阅

[bvp4c](#) | [bvp5c](#) | [bvpinit](#)

详细信息

- “求解边界值问题”（第 12-2 页）

求解具有未知参数的 BVP

以下示例说明如何使用 **bvp4c** 求解具有未知参数的边界值问题。

马蒂厄方程在区间 $[0, \pi]$ 上定义为

$$y'' + (\lambda - 2q \cos(2x))y = 0.$$

当参数 $q = 5$ 时，边界条件为

$$y'(0) = 0,$$

$$y'(\pi) = 0.$$

但这最多只能将 $y(x)$ 确定为一个数乘，因此需要第三个条件来指定特定解，

$$y(0) = 1.$$

要在 MATLAB 中对此方程组求解，您需要先编写方程组、边界条件和初始估计值的代码，然后再调用边界值问题求解器 **bvp4c**。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写方程代码

创建一个函数以编写方程代码。此函数应具有签名 **dydx = mat4ode(x,y,lambda)**，其中：

- **x** 是自变量。
- **y** 是因变量。
- **lambda** 是表示特征值的未知参数。

您可以用代换法 $y_1 = y$ 和 $y_2 = y'$ 将马蒂厄方程写成一阶方程组，

$$y_1' = y_2,$$

$$y_2' = -(\lambda - 2q \cos(2x))y_1.$$

则对应的函数是

```
function dydx = mat4ode(x,y,lambda) % equation being solved
dydx = [y(2)
        -(lambda - 2*q*cos(2*x))*y(1)];
end
```

注意：所有函数都作为局部函数包含在示例的末尾。

编写边界条件代码

现在，编写一个函数，该函数返回在边界点处的边界条件的残差值。此函数应具有签名 **res = mat4bc(ya,yb,lambda)**，其中：

- **ya** 是在区间 $[a, b]$ 开始处的边界条件的值。
- **yb** 是在区间 $[a, b]$ 结束处的边界条件的值。
- **lambda** 是表示特征值的未知参数。

此问题在区间 $[0, \pi]$ 内有三个边界条件。要计算残差值，您需要将边界条件设置为 $g(x, y) = 0$ 形式。在此形式中，边界条件是

$$\begin{aligned}y'(0) &= 0, \\y'(\pi) &= 0, \\y(0) - 1 &= 0.\end{aligned}$$

则对应的函数是

```
function res = mat4bc(ya,yb,lambda) % boundary conditions
res = [ya(2)
yb(2)
ya(1)-1];
end
```

创建初始估计值

最后，创建解的初始估计值。您必须对两个解分量 $y_1 = y(x)$ 和 $y_2 = y'(x)$ 以及未知参数 λ 提供初始估计值。

对于此问题，余弦函数满足三个边界条件，因此有助于提供较好的初始估计值。使用返回 y_1 和 y_2 的估计值的函数，编写 y 的初始估计值的代码。

```
function yinit = mat4init(x) % initial guess function
yinit = [cos(4*x)
-4*sin(4*x)];
end
```

使用区间为 $[0, \pi]$ 的 10 点网格、初始估计值函数以及 λ 的估计值 15 调用 **bvpinit**。

```
lambda = 15;
solinit = bvpinit(linspace(0,pi,10),@mat4init,lambda);
```

求解方程

使用 ODE 函数、边界条件函数和初始估计值调用 **bvp4c**。

```
sol = bvp4c(@mat4ode, @mat4bc, solinit);
```

参数值

打印 **bvp4c** 求得的未知参数 λ 的值。此值是马蒂厄方程的第四个特征值 ($q = 5$)。

```
fprintf('Fourth eigenvalue is approximately %7.3f.\n',...
sol.parameters)
```

Fourth eigenvalue is approximately 17.097.

对解进行绘图

使用 **deval** 计算 **bvp4c** 在区间 $[0, \pi]$ 中的 100 个点处计算的解。

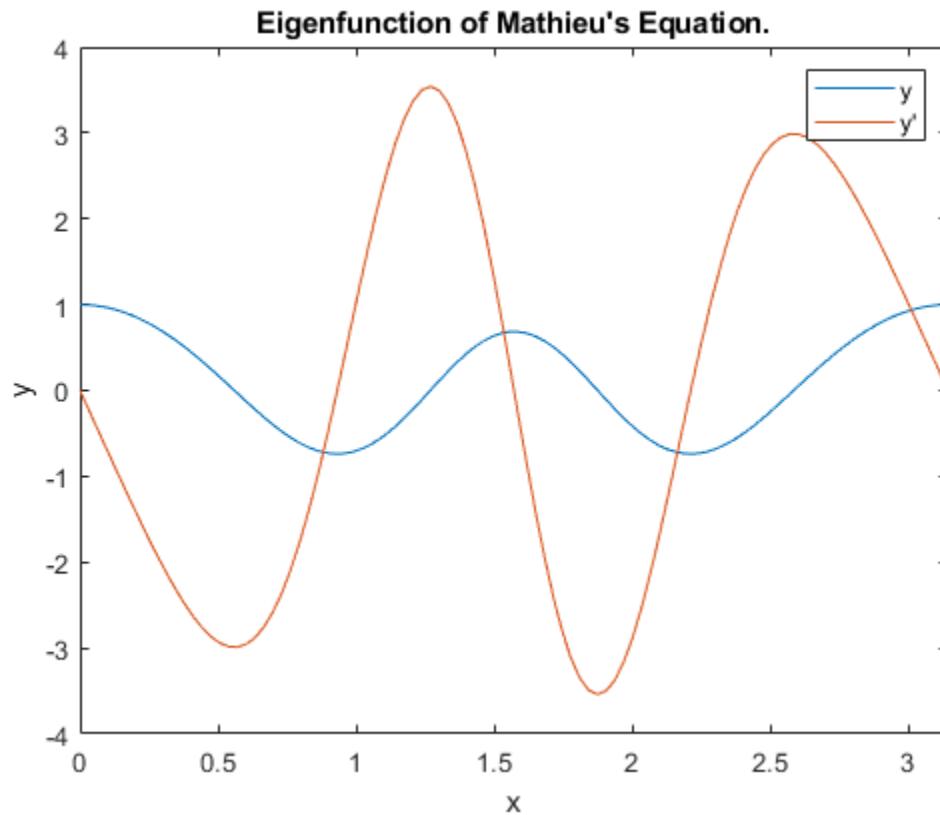
```
xint = linspace(0,pi);
Sxint = deval(sol,xint);
```

对两个解分量进行绘图。绘图显示了与第四个特征值 $\lambda_4 = 17.097$ 相关联的特征函数（及其导数）。

```

plot(xint,Sxint)
axis([0 pi -4 4])
title('Eigenfunction of Mathieu''s Equation.')
xlabel('x')
ylabel('y')
legend('y','y''')

```



局部函数

此处列出了 BVP 求解器 **bvp4c** 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```

function dydx = mat4ode(x,y,lambda) % equation being solved
q = 5;
dydx = [y(2)
        -(lambda - 2*q*cos(2*x))*y(1)];
end
%
function res = mat4bc(ya,yb,lambda) % boundary conditions
res = [ya(2)
       yb(2)
       ya(1)-1];
end
%
function yinit = mat4init(x) % initial guess function
yinit = [cos(4*x)
         -4*sin(4*x)];

```

```
end
```

```
%-----
```

另请参阅

[bvp4c](#) | [bvp5c](#) | [bvpinit](#)

详细信息

- “求解边界值问题” (第 12-2 页)

使用延拓求解 BVP 问题

以下示例说明如何使用延拓求解难以进行数值求解的边界值问题，延拓实际上是将问题分解成一系列更简单的问题。

对于 $0 < e \ll 1$ ，考虑如下微分方程

$$ey'' + xy' = -e\pi^2\cos(\pi x) - \pi x \sin(\pi x).$$

此问题位于区间 $[-1, 1]$ 上，并且需要满足边界条件

$$y(-1) = -2,$$

$$y(1) = 0.$$

当 $e = 10^{-4}$ 时，方程的解会在 $x = 0$ 附近快速转变，因此难以进行数值求解。此示例使用延拓对 e 的几个值进行迭代处理，直到 $e = 10^{-4}$ 。每个中间解都用作下一个问题的初始估计值。

要在 MATLAB 中对此方程组求解，您需要先编写方程组、边界条件和初始估计值的代码，然后再调用边界值问题求解器 **bvp4c**。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），也可以将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写方程代码

使用代换法 $y_1 = y$ 和 $y_2 = y'$ ，您可以将方程重写为一阶方程组

$$y_1' = y_2,$$

$$y_2' = -\frac{x}{e}y' - \pi^2\cos(\pi x) - \frac{\pi x}{e}\sin(\pi x).$$

编写一个函数以使用签名 **dydx = shockode(x,y)** 编写方程代码，其中：

- **x** 是自变量。
- **y** 是因变量。
- **dydx(1)** 给出 y_1' 的方程，**dydx(2)** 给出 y_2' 的方程。

将函数向量化，以使 **shockode([x1 x2 ...],[y1 y2 ...])** 返回 **[shockode(x1,y1) shockode(x2,y2) ...]**。这种方法提高了求解器的性能。

对应的函数是

```
function dydx = shockode(x,y)
pix = pi*x;
dydx = [y(2,:);
-x/e.*y(2,:)-pi^2*cos(pix)-pix/e.*sin(pix)];
end
```

注意：所有函数都作为局部函数包含在示例的末尾。

编写边界条件代码

BVP 求解器要求边界条件采用 $g(y(a), y(b)) = 0$ 形式。在此形式中，边界条件是：

$$y(-1) + 2 = 0,$$

$y(1) = 0$.

编写一个函数以使用签名 `res = shockbc(ya,yb)` 来编写边界条件代码，其中：

- `ya` 是在区间 $[a, b]$ 开始处的边界条件的值。
- `yb` 是在区间 $[a, b]$ 结束处的边界条件的值。

对应的函数是

```
function res = shockbc(ya,yb) % boundary conditions
res = [ya(1)+2
       yb(1)];
end
```

编写 Jacobian 矩阵代码

在此问题中，ODE 函数和边界条件的解析 Jacobian 矩阵可以很轻松地计算出来。提供 Jacobian 矩阵使得求解器效率更高，因为求解器不再需要通过有限差分来逼近它们。

对于 ODE 函数，Jacobian 矩阵为

$$J_{\text{ODE}} = \frac{\partial f}{\partial y} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{x}{e} \end{bmatrix}.$$

对应的函数是

```
function jac = shockjac(x,y,e)
jac = [0 1
       0 -x/e];
end
```

同样，对于边界条件，Jacobian 矩阵为

$$J_{y(a)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad J_{y(b)} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}.$$

对应的函数是

```
function [dBCdy_a,dBCdy_b] = shockbcjac(ya,yb)
dBCdy_a = [1 0; 0 0];
dBCdy_b = [0 0; 1 0];
end
```

获取初始估计值

使用常量估计值在包含 $[-1, 1]$ 中的五个点的网格上求解。

```
sol = bvpinit([-1 -0.5 0 0.5 1],[1 0]);
```

求解方程

如果您尝试使用 $e = 10^{-4}$ 直接求解方程，则求解器会由于问题在转变点 $x = 0$ 附近处的不良条件而难以求解。在这种情况下，为了获得 $e = 10^{-4}$ 的解，此示例使用了延拓，即对 10^{-2} 、 10^{-3} 和 10^{-4} 求解一系列

问题。在每次迭代中求解器的输出充当下一次迭代中解的估计值（这就是为什么 `bvpinit` 的初始估计值的变量是 `sol`, 求解器的输出也命名为 `sol`）。

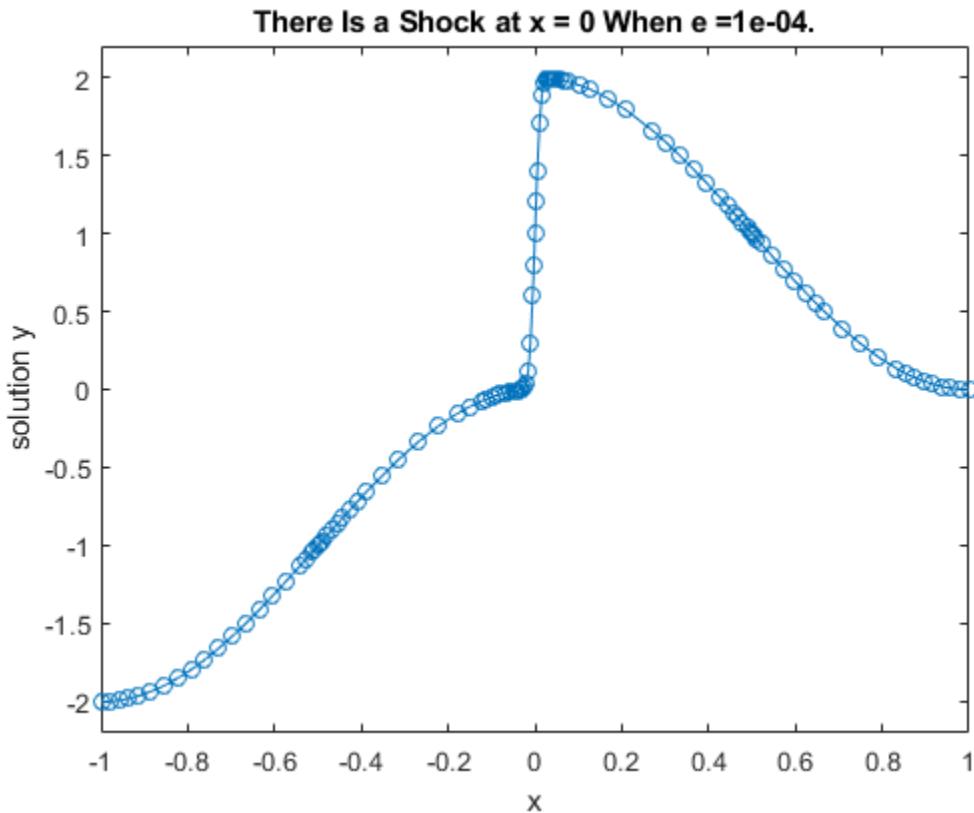
由于 Jacobian 矩阵的值取决于 e 的值，因此需要设置循环中的选项，为 Jacobian 矩阵指定 `shockjac` 和 `shockbcjac` 函数。此外，还要启用向量化，因为编写的 `shockode` 用于处理值向量。

```
e = 0.1;
for i = 2:4
    e = e/10;
    options = bvpset('FJacobian',@(x,y) shockjac(x,y,e),'BCJacobian',@shockbcjac,'Vectorized','on');
    sol = bvp4c(@(x,y) shockode(x,y,e),@shockbc, sol, options);
end
```

对解进行绘图

基于网格 x 和解 $y(x)$ 绘制 `bvp4c` 的输出。使用延拓时，求解器能够处理在 $x = 0$ 处的不连续性。

```
plot(sol.x,sol.y(1,:),'o');
axis([-1 1 -2.2 2.2]);
title(['There Is a Shock at x = 0 When e =' sprintf('%.e',e)' .']);
xlabel('x');
ylabel('solution y');
```



局部函数

此处列出了 BVP 求解器 `bvp4c` 为计算解而调用的局部函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```
function dydx = shockode(x,y,e) % equation to solve
pix = pi*x;
dydx = [y(2,:)
-x/e.*y(2,:)-pi^2*cos(pix)-pix/e.*sin(pix)];
end
%
function res = shockbc(ya,yb) % boundary conditions
res = [ya(1)+2
yb(1)];
end
%
function jac = shockjac(x,y,e) % jacobian of shockode
jac = [0 1
0 -x/e];
end
%
function [dBCdyA,dBCdyB] = shockbcjac(ya,yb) % jacobian of shockbc
dBCdyA = [1 0; 0 0];
dBCdyB = [0 0; 1 0];
end
%
```

另请参阅

[bvp4c](#) | [bvpinit](#) | [bvpset](#)

详细信息

- “求解边界值问题” (第 12-2 页)
- “使用延拓验证 BVP 一致性” (第 12-18 页)

使用延拓验证 BVP 一致性

以下示例说明如何使用延拓将 BVP 的一个解逐渐扩展到更大的区间。

Falkner-Skan 边界值问题 [1] 源于为平板粘性不可压缩层流问题求取相似解的过程。示例方程是

$$f''' + f f'' + \beta(1 - f^2) = 0.$$

此问题位于无限区间 $[0, \infty]$ 和 $\beta = 0.5$ 上，并且需要满足边界条件

$$f(0) = 0,$$

$$f'(0) = 0,$$

$$f'(\infty) = 1.$$

在无限区间上无法求解 BVP，在非常大的有限区间上求解 BVP 也不切实际。在这种情况下，此示例转而求解位于较小区间 $[0, a]$ 上的一系列问题，从而验证解具有与 $a \rightarrow \infty$ 一致的行为。这种做法称为延拓，它是将一个问题分解成多个更简单的问题，将每个小问题所反馈的解作为下一个大问题的初始估计值。

要在 MATLAB 中对此方程组求解，您需要先编写方程组、边界条件和选项的代码，然后再调用边界值问题求解器 **bvp4c**。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写方程代码

创建一个函数以编写方程代码。此函数应具有签名 **dfdx = fsode(x,f)**，其中：

- x 是自变量。
- f 是因变量。

您可以使用代换法 $f_1 = f$ 、 $f_2 = f'$ 和 $f_3 = f''$ 将三阶方程重写为一阶方程组。方程变为

$$f_1' = f_2,$$

$$f_2' = f_3,$$

$$f_3' = -f_1 f_3 - \beta(1 - f_2^2).$$

对应的函数是

```
function dfdata = fsode(x,f)
b = 0.5;
dfdata = [ f(2)
            f(3)
            -f(1)*f(3) - b*(1 - f(2)^2) ];
end
```

注意：所有函数都作为局部函数包含在示例的末尾。

编写边界条件代码

现在，编写一个函数，该函数返回在边界点处的边界条件的残差值。此函数应具有签名 **res = fsbc(f0,finf)**，其中：

- **f0** 是在区间的开始处的边界条件的值。
- **finf** 是在区间的结束处的边界条件的值。

要计算残差值，您需要将边界条件设置为 $g(x, y) = 0$ 形式。在此形式中，边界条件是

$$f(0) = 0,$$

$$f'(0) = 0,$$

$$f'(\infty) - 1 = 0.$$

对应的函数是

```
function res = fsbc(f0,finf)
res = [f0(1)
       f0(2)
       finf(2) - 1];
end
```

创建初始估计值

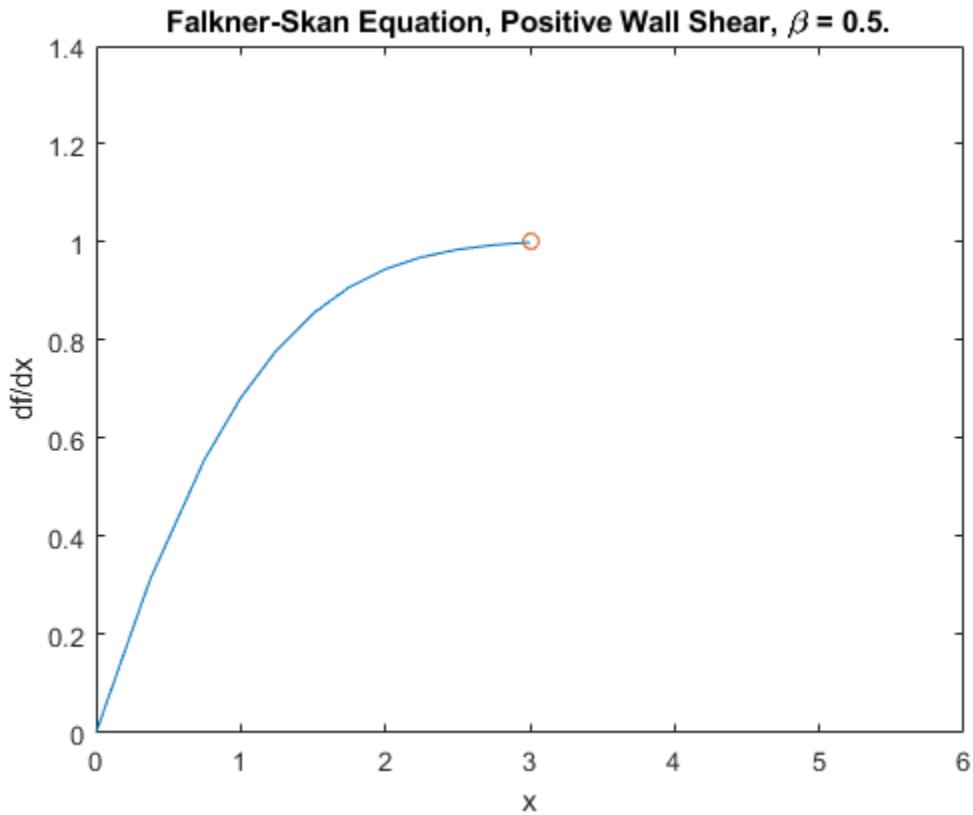
最后，您必须提供解的初始估计值。具有五个点的粗网格以及满足边界条件的常量估计值便可在区间 $[0, 3]$ 上进行收敛。变量 **infinity** 表示积分区间的右侧极限。随着 **infinity** 的值在随后的迭代中从 3 增加到其最大值 6，每个先前迭代给出的解充当下一次迭代的初始估计值。

```
infinity = 3;
maxinfinity = 6;
solinit = bvpinit(linspace(0,infinity,5),[0 0 1]);
```

求解方程并绘制解

在初始区间 $[0, 3]$ 中求解问题。绘制解，并将 $f''(0)$ 的值与解析值 [1] 进行比较。

```
sol = bvp4c(@fsode,@fsbc,solinit);
x = sol.x;
f = sol.y;
plot(x,f(2,:),x(end),f(2,end),'o');
axis([0 maxinfinity 0 1.4]);
title('Falkner-Skan Equation, Positive Wall Shear, \beta = 0.5.')
xlabel('x')
ylabel('df/dx')
hold on
```



```
fprintf('Cebeci & Keller report that f'''(0) = 0.92768.\n')
```

Cebeci & Keller report that $f'(0) = 0.92768$.

```
fprintf('Value computed using infinity = %g is %7.5f.\n', ...
infinity,f(3,1))
```

Value computed using infinity = 3 is 0.92915.

现在，通过为每次迭代增加 `infinity` 的值，在逐步增大的区间上求解问题。`bvpinit` 函数将每个解外推至新区间，以作为 `infinity` 的下一个值的初始估计值。每次迭代都会打印 $f''(0)$ 的计算值，并将解的绘图叠加到之前的解上。当 `infinity = 6` 时，解的一致行为变得明显， $f''(0)$ 的值非常接近预测值。

```
for Bnew = infinity+1:maxinfinity
solinit = bvpinit(sol,[0 Bnew]); % Extend solution to new interval
sol = bvp4c(@fsode,@fsbc,solinit);
x = sol.x;
f = sol.y;

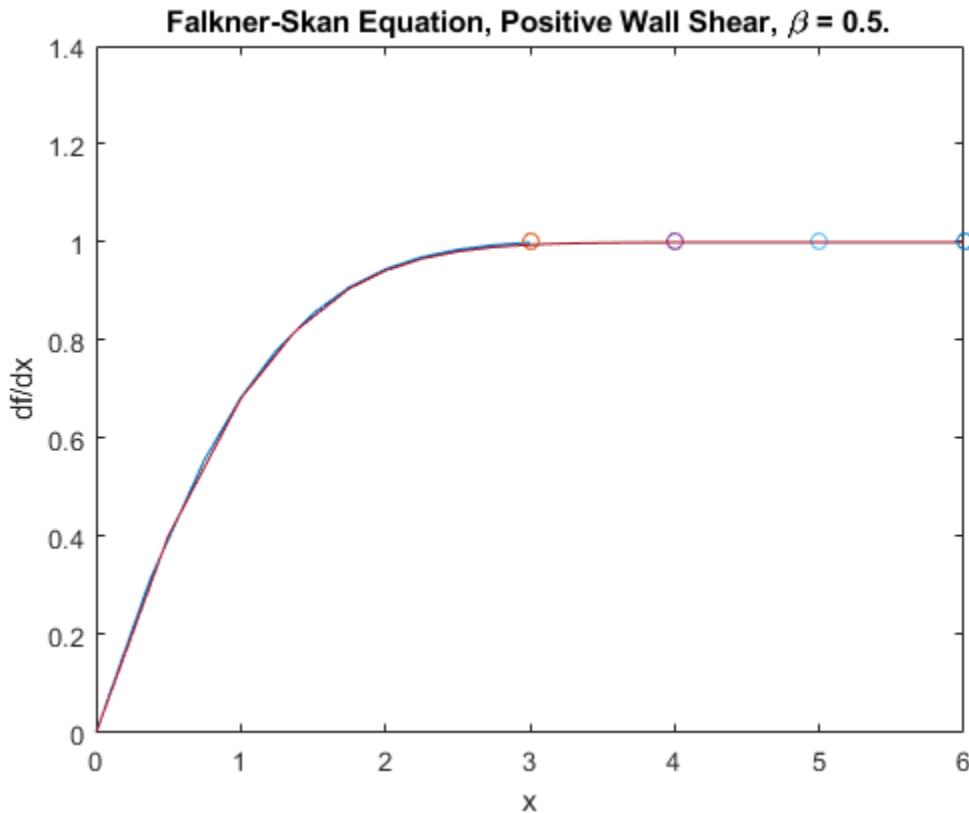
fprintf('Value computed using infinity = %g is %7.5f.\n', ...
Bnew,f(3,1))
plot(x,f(2,:),x(end),f(2,end),'o');
drawnow
end
```

Value computed using infinity = 4 is 0.92774.

Value computed using infinity = 5 is 0.92770.

Value computed using infinity = 6 is 0.92770.

hold off



局部函数

此处列出了 BVP 求解器 **bvp4c** 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```
function dfdata = fsode(x,f) % equation being solved
dfdata = [ f(2)
           f(3)
           -f(1)*f(3) - 0.5*(1 - f(2)^2) ];
end
%
function res = fsbc(f0,finf) % boundary conditions
res = [f0(1)
       f0(2)
       finf(2) - 1];
end
%
```

参考

[1] Cebeci, T. and H. B. Keller."Shooting and Parallel Shooting Methods for Solving the Falkner-Skan Boundary-layer Equation."J. Comp.Phys., Vol. 7, 1971, pp. 289-300.

另请参阅

[bvp4c](#) | [bvp5c](#) | [bvpinit](#)

详细信息

- “求解边界值问题” (第 12-2 页)
- “使用延拓求解 BVP 问题” (第 12-14 页)

求解具有奇异项的 BVP

以下示例说明如何求解埃姆登方程，埃姆登方程是一个具有奇异项的边界值问题，源于对气体球体建模的过程。

在使用对称性法简化模型的 PDE 后，该方程变为在区间 $[0, 1]$ 上定义的二阶 ODE，

$$y'' + \frac{2}{x}y' + y^5 = 0.$$

在 $x = 0$ 处， $(2/x)$ 项具有奇异性，但对称性表示边界条件 $y'(0) = 0$ 。通过此边界条件，项 $(2/x)y'$ 可以很好地定义为 $x \rightarrow 0$ 。对于边界条件 $y(1) = \sqrt{3}/2$ ，BVP 有解析解，可以将该解与 MATLAB® 中计算的数值解进行比较，

$$y(x) = \left[\sqrt{\frac{1+x^2}{3}} \right]^{-1}.$$

BVP 求解器 **bvp4c** 可以求解以下形式的奇异 BVP

$$y' = S \frac{y}{x} + f(x, y).$$

矩阵 S 必须为常量， $x = 0$ 处的边界条件必须与必要条件 $S \cdot y(0) = 0$ 一致。使用 **bvpset** 的 'SingularTerm' 选项将 S 矩阵传递给求解器。

您可以使用 $y_1 = y$ 和 $y_2 = y'$ 将埃姆登方程重写为一阶方程组

$$y_1' = y_2,$$

$$y_2' = -\frac{2}{x}y_2 - y_1^5.$$

边界条件变为

$$y_2(0) = 0,$$

$$y_1(1) = \sqrt{3}/2.$$

采用要求的矩阵形式，方程组表示为

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \frac{1}{x} \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} y_2 \\ -y_1^5 \end{bmatrix}.$$

$$\text{采用矩阵形式时，很明显，} S = \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix} \text{ 且 } f(x, y) = \begin{bmatrix} y_2 \\ -y_1^5 \end{bmatrix}.$$

要在 MATLAB 中对此方程组求解，您需要先编写方程组、边界条件和选项的代码，然后再调用边界值问题求解器 **bvp4c**。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写方程代码

创建一个函数以用于编写 $f(x, y)$ 的方程代码。此函数应具有签名 `dydx = emdenode(x,y)`，其中：

- x 是自变量。
- y 是因变量。
- $\text{dydx}(1)$ 给出 y_1' 的方程, $\text{dydx}(2)$ 给出 y_2' 的方程。

求解器会自动将这些输入传递给该函数, 但是变量名称决定如何编写方程代码。在这种情况下:

```
function dydx = emdenode(x,y)
dydx = [y(2)
        -y(1)^5];
end
```

包含 S 的项通过选项传递给求解器, 因此该项不包含在函数中。

编写边界条件代码

现在, 编写一个函数, 该函数返回在边界点处的边界条件的残差值。此函数应具有签名 $\text{res} = \text{emdenbc}(ya,yb)$, 其中:

- ya 是在区间的开始处的边界条件的值。
- yb 是在区间的结束处的边界条件的值。

对于此问题, 一个边界条件针对 y_1 , 另一个边界条件针对 y_2 。要计算残差值, 您需要将边界条件设置为 $g(x, y) = 0$ 形式。

在此形式中, 边界条件是

$$y_2(0) = 0,$$

$$y_1(1) - \sqrt{3}/2 = 0.$$

则对应的函数是

```
function res = emdenbc(ya,yb)
res = [ya(2)
       yb(1) - sqrt(3)/2];
end
```

创建初始估计值

最后, 创建解的初始估计值。对于此问题, 使用满足边界条件的常量初始估计值, 以及包含介于 0 和 1 之间的五个点的简单网格。没有必要使用许多网格点, 因为 BVP 求解器会在求解过程中调整这些点。

$$y_1 = \sqrt{3}/2,$$

$$y_2 = 0.$$

```
guess = [sqrt(3)/2; 0];
xmesh = linspace(0,1,5);
solinit = bvpinit(xmesh, guess);
```

求解方程

为 S 创建矩阵, 并将其作为 'SingularTerm' 选项的值传递给 bvpset 。最后, 使用 ODE 函数、边界条件函数、初始估计值和 options 结构体调用 bvp4c 。

```
S = [0 0; 0 -2];
options = bvpset('SingularTerm',S);
sol = bvp4c(@emdenode, @emdenbc, solinit, options);
```

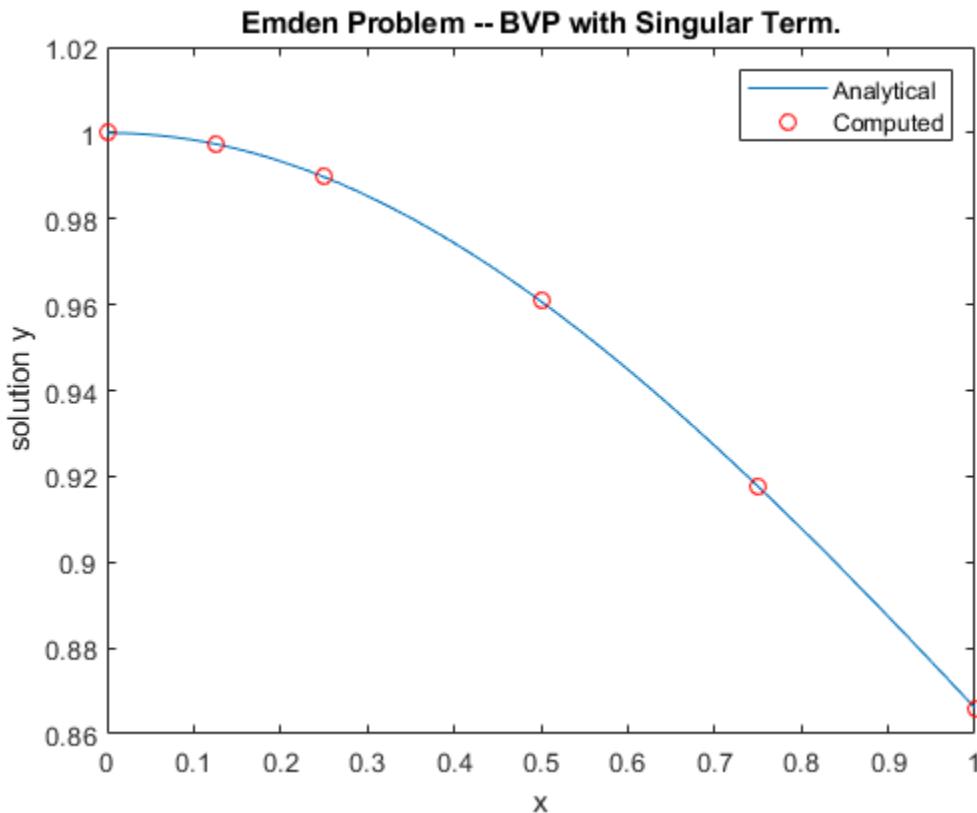
对解进行绘图

计算 $[0, 1]$ 中解析解的值。

```
x = linspace(0,1);
truy = 1 ./ sqrt(1 + (x.^2)/3);
```

绘制解析解和 bvp4c 计算的解，以进行比较。

```
plot(x,truy,sol.x,sol.y(1,:),'ro');
title('Emden Problem -- BVP with Singular Term.')
legend('Analytical','Computed');
xlabel('x');
ylabel('solution y');
```



局部函数

此处列出了 BVP 求解器 bvp4c 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```
function dydx = emdenode(x,y) % equation being solved
dydx = [y(2)
        -y(1)^5];
end
```

```
%-----  
function res = emdenbc(ya,yb) % boundary conditions  
res = [ya(2)  
       yb(1) - sqrt(3)/2];  
end  
%-----
```

另请参阅

[bvp4c](#) | [bvp5c](#) | [bvpinit](#) | [bvpset](#)

详细信息

- “求解边界值问题” (第 12-2 页)

求解具有多边界条件的 BVP

以下示例说明如何求解多点边界值问题，其中关注的解满足积分区间内的条件。

对于 $[0, \lambda]$ 中的 x ，考虑以下方程

$$v' = \frac{C - 1}{n},$$

$$C' = \frac{vC - \min(x, 1)}{\eta}.$$

问题的已知参数是 n 、 κ 、 $\lambda > 1$ 和 $\eta = \frac{\lambda^2}{n \cdot \kappa^2}$ 。

$C(x)$ 的方程中的项 $\min(x, 1)$ 在 $x = 1$ 处不平滑，因此该问题不能直接求解。在这种情况下，您可以将问题分成两部分：一部分在区间 $[0, 1]$ 内，另一部分在区间 $[1, \lambda]$ 内。这两个区域之间的联系是在 $x = 1$ 处的解必须为连续的。解还必须满足边界条件

$$v(0) = 0,$$

$$C(\lambda) = 1.$$

每个区域的方程如下

区域 1： $0 \leq x \leq 1$

$$v' = \frac{C - 1}{n},$$

$$C' = \frac{vC - x}{\eta}.$$

区域 2： $1 \leq x \leq \lambda$

$$v' = \frac{C - 1}{n},$$

$$C' = \frac{vC - 1}{\eta}.$$

交界点 $x = 1$ 同时包含在这两个区域中。在此交界点上，求解器会产生左解和右解，这两个解必须相等，以确保解的连续性。

要在 MATLAB 中对此方程组求解，您需要先编写方程组、边界条件和初始估计值的代码，然后再调用边界值问题求解器 **bvp5c**。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写方程代码

$v'(x)$ 和 $C'(x)$ 的方程取决于正在求解的区域。对于多点边界值问题，导数函数必须接受第三个输入参数 **region**，该参数用于标识正在计算导数的区域。求解器从左到右对区域进行编号，从 1 开始。

创建一个函数以使用签名 **dydx = f(x,y,region,p)** 来编写方程代码，其中：

- x 是自变量。

- y 是因变量。
- $\text{dydx}(1)$ 给出 $v'(x)$ 的方程, $\text{dydx}(2)$ 给出 $C'(x)$ 的方程。
- **region** 是计算导数的区域编号 (本例中问题分为两个区域, **region** 为 1 或 2)。
- **p** 是向量, 包含常量参数 $[n, \kappa, \lambda, \eta]$ 的值。

根据要求解的具体区域, 使用 switch 语句返回不同方程。函数是

```
function dydx = f(x,y,region,p) % equations being solved
n = p(1);
eta = p(4);

dydx = zeros(2,1);
dydx(1) = (y(2) - 1)/n;

switch region
    case 1 % x in [0 1]
        dydx(2) = (y(1)*y(2) - x)/eta;
    case 2 % x in [1 lambda]
        dydx(2) = (y(1)*y(2) - 1)/eta;
end
end
```

注意: 所有函数都作为局部函数包含在示例的末尾。

编写边界条件代码

在两个区域中求解两个一阶微分方程需要四个边界条件。这些条件中有两个来自原始问题:

$$v(0) = 0,$$

$$C(\lambda) - 1 = 0.$$

另外两个条件强制交界点 $x = 1$ 处的左解和右解具备连续性:

$$v_L(1) - v_R(1) = 0,$$

$$C_L(1) - C_R(1) = 0.$$

对于多点 BVP, 边界条件函数 **YL** 和 **YR** 的参数会是矩阵。具体来说, 第 **k** 列 **YL(:,k)** 是第 **k** 个区域左边界的解。**YR(:,k)** 则是第 **k** 个区域右边界的解。

在此问题中, $y(0)$ 通过 **YL(:,1)** 来逼近, 而 $y(\lambda)$ 通过 **YR(:,end)** 来逼近。解在 $x = 1$ 处的连续性要求 **YR(:,1) = YL(:,2)**。

用于编写四个边界条件的残差值计算代码的函数是

```
function res = bc(YL,YR)
res = [YL(1,1) % v(0) = 0
       YR(1,1) - YL(1,2) % Continuity of v(x) at x=1
       YR(2,1) - YL(2,2) % Continuity of C(x) at x=1
       YR(2,end) - 1]; % C(lambda) = 1
end
```

获取初始估计值

对于多点 BVP，边界条件自动应用于积分区间的开始处和结束处。但是，您必须在 `xmesh` 中为其他交界点分别指定双重项。满足边界条件的简单估计值是常量估计值 $y = [1; 1]$ 。

```
xc = 1;
xmesh = [0 0.25 0.5 0.75 xc xc 1.25 1.5 1.75 2];
yinit = [1; 1];
sol = bvpinit(xmesh,yinit);
```

求解方程

定义常量参数的值，并将其放入向量 `p` 中。使用语法 `@(x,y,r) f(x,y,r,p)` 向 `bvp5c` 提供函数，以提供参数向量。

计算 κ 的几个值的解，其中使用每个解作为下一个解的初始估计值。对于 κ 的每个值，计算渗透性 $O_s = \frac{1}{v(\lambda)}$ 的值。对于循环的每次迭代，将计算值与近似解析解进行比较。

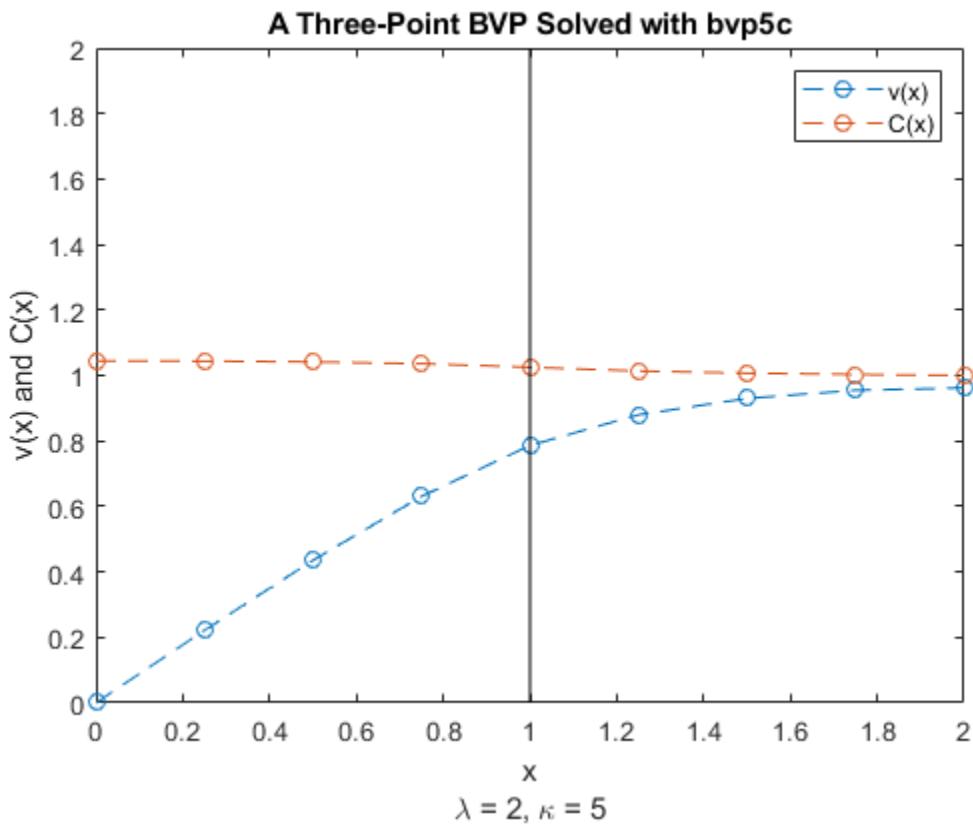
```
lambda = 2;
n = 5e-2;
for kappa = 2:5
    eta = lambda^2/(n*kappa^2);
    p = [n kappa lambda eta];
    sol = bvp5c(@(x,y,r) f(x,y,r,p), @bc, sol);
    K2 = lambda*sinh(kappa/lambda)/(kappa*cosh(kappa));
    approx = 1/(1 - K2);
    computed = 1/sol.y(1,end);
    fprintf(' %2i %10.3f %10.3f\n',kappa,computed,approx);
end
```

2	1.462	1.454
3	1.172	1.164
4	1.078	1.071
5	1.039	1.034

对解进行绘图

绘制 $v(x)$ 和 $C(x)$ 的解分量，以及在交界点 $x = 1$ 处的垂直线。显示的 $\kappa = 5$ 的解是循环的最后一次迭代的结果。

```
plot(sol.x,sol.y(1,:),'-o',sol.x,sol.y(2,:),'-o')
line([1 1], [0 2], 'Color', 'k')
legend('v(x)', 'C(x)')
title('A Three-Point BVP Solved with bvp5c')
xlabel({'x', '\lambda = 2, \kappa = 5'})
ylabel('v(x) and C(x)')
```



局部函数

此处列出了 BVP 求解器 **bvp5c** 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```

function dydx = f(x,y,region,p) % equations being solved
n = p(1);
eta = p(4);

dydx = zeros(2,1);
dydx(1) = (y(2) - 1)/n;

switch region
    case 1 % x in [0 1]
        dydx(2) = (y(1)*y(2) - x)/eta;
    case 2 % x in [1 lambda]
        dydx(2) = (y(1)*y(2) - 1)/eta;
end
end
%
function res = bc(YL,YR) % boundary conditions
res = [YL(1,1) % v(0) = 0
        YR(1,1) - YL(1,2) % Continuity of v(x) at x=1
        YR(2,1) - YL(2,2) % Continuity of C(x) at x=1
        YR(2,end) - 1]; % C(lambda) = 1

```

end

%-----

另请参阅

[bvp4c](#) | [bvp5c](#) | [bvpinit](#)

详细信息

- “求解边界值问题” (第 12-2 页)

偏微分方程 (PDE)

- “求解偏微分方程” (第 13-2 页)
- “求解单个 PDE” (第 13-9 页)
- “求解具有不连续性的 PDE” (第 13-16 页)
- “求解 PDE 并计算偏导数” (第 13-22 页)
- “求解 PDE 方程组” (第 13-30 页)
- “使用初始条件阶跃函数求解 PDE 方程组” (第 13-37 页)

求解偏微分方程

在偏微分方程 (PDE) 中，要求解的函数取决于几个变量，微分方程可以包括关于每个变量的偏导数。偏微分方程可用于对波浪、热流、流体扩散和其他空间行为随时间变化的现象建模。

使用 MATLAB 可求解哪些类型的 PDE?

MATLAB PDE 求解器 **pdepe** 使用一个空间变量 x 和时间 t 对 PDE 方程组的初始边界值问题求解。您可以将这些看作一个变量的 ODE，它们也会随着时间而变化。

pdepe 要求解的一维方程大概可分为以下两类：

- 带时间导数的方程是抛物型方程。例如热方程 $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$ 。
- 不带时间导数的方程是椭圆型方程。例如拉普拉斯方程 $\frac{\partial^2 u}{\partial x^2} = 0$ 。

pdepe 要求方程组中存在至少一个抛物型方程。换句话说，方程组中至少一个方程必须包含时间导数。

pdepe 还可求解某些二维和三维问题，这些问题由于角对称而简化为一维问题（有关详细信息，请参阅对称常量 **m** 的参数说明）。

Partial Differential Equation Toolbox 将此功能扩展到 Dirichlet 和 Neumann 边界条件下的二维和三维广义问题。

求解一维 PDE

一维 PDE 包含函数 $u(x,t)$ ，该函数依赖于时间 t 和一个空间变量 x 。MATLAB PDE 求解器 **pdepe** 求解以下形式的一维抛物型和椭圆型 PDE 的方程组

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x} \left(x^m f\left(x, t, u, \frac{\partial u}{\partial x}\right) \right) + s\left(x, t, u, \frac{\partial u}{\partial x}\right).$$

方程具有以下属性：

- PDE 在 $t_0 \leq t \leq t_f$ 和 $a \leq x \leq b$ 时成立。
- 空间区间 $[a, b]$ 必须为有限值。
- m 可以是 0、1 或 2，分别对应平板、柱状或球面对称性。如果 $m > 0$ ，则 $a \geq 0$ 也必须成立。
- 系数 $f\left(x, t, u, \frac{\partial u}{\partial x}\right)$ 是通量项， $s\left(x, t, u, \frac{\partial u}{\partial x}\right)$ 是源项。
- 通量项必须取决于偏导数 $\partial u / \partial x$ 。

关于时间的偏导数耦合只限于与对角矩阵 $c\left(x, t, u, \frac{\partial u}{\partial x}\right)$ 相乘。此矩阵的对角线元素为零或正数。为零的元素对应于椭圆型方程，任何其他元素对应于抛物型方程。必须至少存在一个抛物型方程。如果 x 的某些孤立值是网格点（即计算解的位置），那么在这些值处，抛物型方程对应的 c 元素可能消失。当物质界面上有网格点时，允许 c 和 s 中出现界面导致的不连续点。

求解过程

要使用 **pdepe** 求解 PDE，您必须定义 **c**、**f** 和 **s** 的方程系数、初始条件、解在边界处的行为以及在其上计算解的点网格。函数调用 **sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan)** 使用以下信息计算指定网格上的一个解：

- **m** 是对称常量。
- **pdefun** 定义要求解的方程。
- **icfun** 定义初始条件。
- **bcfun** 定义边界条件。
- **xmesh** 是 **x** 的空间值向量。
- **tspan** 是 **t** 的时间值向量。

xmesh 和 **tspan** 向量共同构成一个二维网格，**pdepe** 在该网格上计算解。

方程

您必须按照 **pdepe** 所需的标准形式表示 PDE。以这种形式编写，您可以读取系数 **c**、**f**、**s** 的值。

在 MATLAB 中，您可以用以下形式的函数编写方程代码

```
function [c,f,s] = pdefun(x,t,u,dudx)
c = 1;
f = dudx;
s = 0;
end
```

在本例中，**pdefun** 定义方程 $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$ 。如果有多个方程，则 **c**、**f** 和 **s** 均为向量，其中每个元素对应于一个方程。

初始条件

在初始时间 $t = t_0$ 时，针对所有 x ，解分量均满足以下格式的初始条件

$$u(x, t_0) = u_0(x).$$

在 MATLAB 中，您可以用以下形式的函数对初始条件进行编码

```
function u0 = icfun(x)
u0 = 1;
end
```

在本例中，**u0 = 1** 定义 $u_0(x, t_0) = 1$ 的初始条件。如果有多个方程，则 **u0** 是一个向量，其中每个元素定义一个方程的初始条件。

边界条件

在边界 $x = a$ 或 $x = b$ 时，针对所有 t ，解分量满足以下形式的边界条件

$$p(x, t, u) + q(x, t)f\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0.$$

q(x,t) 是对角线矩阵，其元素全部是零或全部是非零。请注意，边界条件以通量 **f**（而非关于 **x** 的 **u** 的偏导数）形式表示。同时，在 **p(x,t,u)** 和 **q(x,t)** 这两个系数之间，只有 **p** 可以依赖于 **u**。

在 MATLAB 中，您可以用以下形式的函数对边界条件进行编码

```
function [pL,qL,pR,qR] = bcfun(xL,uL,xR,uR,t)
pL = uL;
qL = 0;
pR = uR - 1;
qR = 0;
end
```

pL 和 **qL** 是左边界上的系数，**pR** 和 **qR** 是右边界上的系数。在本例中，**bcfun** 定义边界条件

$$u_L(x_L, t) = 0$$

$$u_R(x_R, t) = 1$$

如果有多个方程，则输出 **pL**、**qL**、**pR** 和 **qR** 是向量，其中每个元素定义一个方程的边界条件。

积分选项

可以选择 MATLAB PDE 求解器中的默认积分属性来处理常见问题。在某些情况下，可以通过覆盖这些默认值来提高求解器的性能。为此，请使用 **odeset** 创建一个 **options** 结构体。然后，将该结构体作为最后一个输入参数传递给 **pdepe**：

```
sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan,options)
```

在基础 ODE 求解器 **ode15s** 的选项中，只有下表中所示的选项可用于 **pdepe**。

类别	选项名称
误差控制	RelTol , AbsTol , NormControl
步长大小	InitialStep , MaxStep
事件日志记录	Events

解的计算

在您用 **pdepe** 求解方程后，MATLAB 将以三维数组 **sol** 返回解，其中 **sol(i,j,k)** 包含在 **t(i)** 和 **x(j)** 处计算的解的第 **k** 个分量。通常，您可以使用命令 **u = sol(:,:,k)** 提取第 **k** 个解分量。

您指定的时间网格仅用于输出目的，不影响求解器采用的内部时间步。但是，您指定的空间网格会影响解的质量和速度。求解方程后，您可以使用 **pdeval** 计算 **pdepe** 采用不同空间网格返回的解结构体。

示例：热方程

抛物型 PDE 的一个示例是一维热方程：

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}.$$

此方程描述 $0 \leq x \leq L$ 和 $t \geq 0$ 的散热情况。目标是求解 $u(x, t)$ 温度问题。温度最初是一个非零常量，因此初始条件是

$$u(x, 0) = T_0.$$

此外，左边界上的温度为零，右边界上的温度不为零，因此边界条件为

$$\begin{aligned} u(0, t) &= 0, \\ u(L, t) &= 1. \end{aligned}$$

要在 MATLAB 中求解该方程，您需要对方程、初始条件和边界条件编写代码，然后在调用求解器 **pdepe** 之前选择合适的解网格。您可以将所需的函数作为局部函数包含在文件末尾（如本示例所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写方程代码

在编写方程代码之前，您需要确保它的形式符合 **pdepe** 求解器的要求：

$$c(x, t, u, \frac{\partial u}{\partial t}) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x} \left(x^m f(x, t, u, \frac{\partial u}{\partial x}) \right) + s(x, t, u, \frac{\partial u}{\partial x}).$$

在此形式中，热方程是

$$1 \cdot \frac{\partial u}{\partial t} = x^0 \frac{\partial}{\partial x} \left(x^0 \frac{\partial u}{\partial x} \right) + 0.$$

因此，系数的值如下：

- $m = 0$
- $c = 1$
- $f = \frac{\partial u}{\partial x}$
- $s = 0$

m 的值作为参数传递给 **pdepe**，而其他系数编写为方程的一个函数，即

```
function [c,f,s] = heatpde(x,t,u,dudx)
c = 1;
f = dudx;
s = 0;
end
```

（注意：所有函数都作为局部函数包含在示例的末尾。）

代码初始条件

热方程的初始条件函数对 u_0 赋给一个常量值。此函数必须接受 x 的输入，即使它未使用。

```
function u0 = heatic(x)
u0 = 0.5;
end
```

编写边界条件代码

pdepe 求解器所需的边界条件的标准形式是

$$p(x, t, u) + q(x, t)f\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0.$$

以这种形式编写的此问题的边界条件是

$$u(0, t) + (0 \cdot f) = 0,$$

$$(u(L, t) - 1) + (0 \cdot f) = 0.$$

因此, p 和 q 的值是

- $p_L = u_L, \quad q_L = 0.$
- $p_R = u_R - 1, \quad q_R = 0.$

则对应的函数是

```
function [pl,ql,pr,qr] = heatbc(xl,ul,xr,ur,t)
pl = ul;
ql = 0;
pr = ur - 1;
qr = 0;
end
```

选择解网格

使用包含 20 个点的空间网格和包含 30 个点的时间网格。由于解快速达到稳态, $t = 0$ 附近的时间点间隔更近以将此行为捕获到输出中。

```
L = 1;
x = linspace(0,L,20);
t = [linspace(0,0.05,20), linspace(0.5,5,10)];
```

求解方程

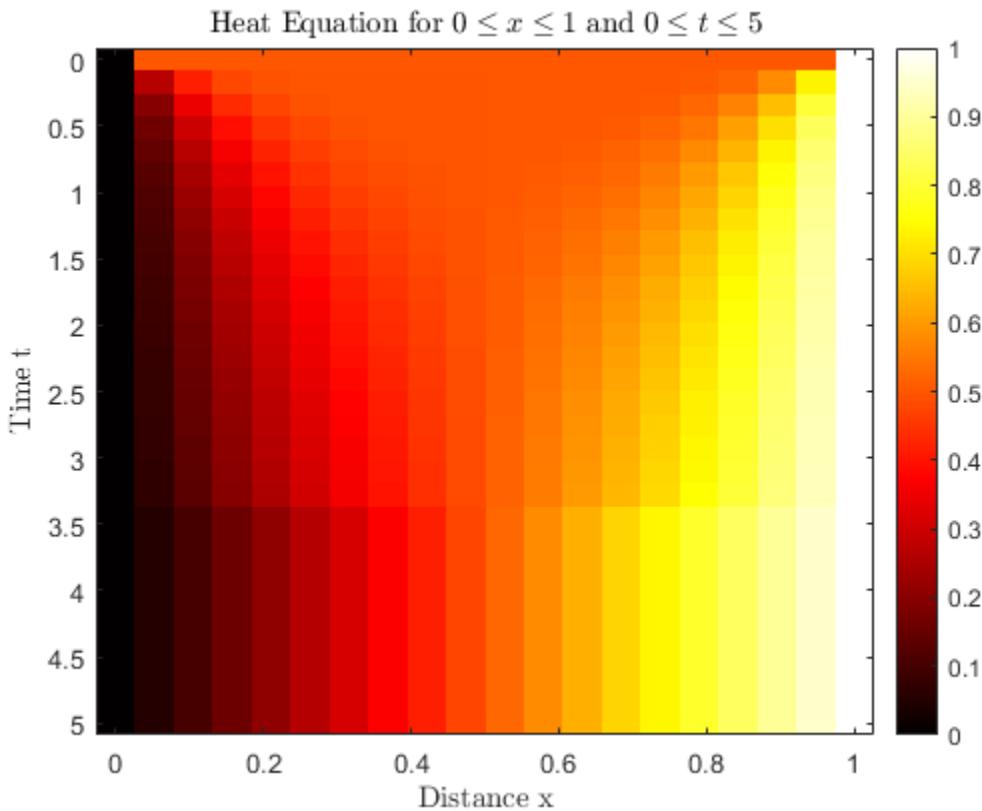
最后, 使用对称性值 m 、PDE 方程、初始条件、边界条件以及 x 和 t 的网格来求解方程。

```
m = 0;
sol = pdepe(m,@heatpde,@heatic,@heatbc,x,t);
```

对解进行绘图

使用 `imagesc` 可视化解矩阵。

```
colormap hot
imagesc(x,t,sol)
colorbar
xlabel('Distance x','interpreter','latex')
ylabel('Time t','interpreter','latex')
title('Heat Equation for $0 \leq x \leq 1$ and $0 \leq t \leq 5$','interpreter','latex')
```



局部函数

```

function [c,f,s] = heatpde(x,t,u,dudx)
c = 1;
f = dudx;
s = 0;
end
function u0 = heatic(x)
u0 = 0.5;
end
function [pl,ql,pr,qr] = heatbc(xl,ul,xr,ur,t)
pl = ul;
ql = 0;
pr = ur - 1;
qr = 0;
end

```

PDE 示例和文件

我们提供几个示例文件，它们可作为求解最常见的一维 PDE 问题的绝佳起点。要查看和运行示例，请使用 Differential Equations Examples App。要运行此 App，请键入

`odeexamples`

要打开单独的文件进行编辑，请键入

`edit exampleFileName.m`

要运行示例，请键入

`exampleFileName`

下表包含可用 PDE 示例文件的列表。

示例文件	说明	示例链接
<code>pdex1</code>	简单的 PDE，用于说明解的公式、计算和绘图。	“求解单个 PDE”（第 13-9 页）
<code>pdex2</code>	涉及不连续性的问题。	“求解具有不连续性的 PDE”（第 13-16 页）
<code>pdex3</code>	需要计算偏导数的值的问题。	“求解 PDE 并计算偏导数”（第 13-22 页）
<code>pdex4</code>	含两个 PDE 的方程组，其解在区间两端具有边界层，并且对于较小的 t 值，解的变化很快。	“求解 PDE 方程组”（第 13-30 页）
<code>pdex5</code>	阶跃函数为初始条件的 PDE 方程组。	“使用初始条件阶跃函数求解 PDE 方程组”（第 13-37 页）

参考

- [1] Skeel, R. D. and M. Berzins, "A Method for the Spatial Discretization of Parabolic Equations in One Space Variable," *SIAM Journal on Scientific and Statistical Computing*, Vol. 11, 1990, pp. 1-32.

另请参阅

`bvp4c` | `ode45` | `odeset` | `pdepe` | `pdeval`

详细信息

- “求解单个 PDE”（第 13-9 页）
- “求解 PDE 方程组”（第 13-30 页）

求解单个 PDE

此示例说明单个 PDE 的解的构成以及如何对解进行计算和绘图。

以如下偏微分方程为例

$$\pi^2 \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}.$$

该方程的定义区间为 $0 \leq x \leq 1$, 时间 $t \geq 0$ 。在 $t = 0$ 时, 解满足初始条件

$$u(x, 0) = \sin(\pi x).$$

此外, 在 $x = 0$ 和 $x = 1$ 时, 解满足边界条件

$$u(0, t) = 0,$$

$$\pi e^{-t} + \frac{\partial u}{\partial x}(1, t) = 0.$$

要在 MATLAB 中求解该方程, 您需要对方程、初始条件和边界条件编写代码, 然后在调用求解器 **pdepe** 之前选择合适的解网格。您可以将所需的函数作为局部函数包含在文件末尾 (如本处所示), 或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写方程代码

在编写方程代码之前, 您需要按照 **pdepe** 求解器所需的形式对其进行重写。**pdepe** 所需的标准形式是

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x} \left(x^m f\left(x, t, u, \frac{\partial u}{\partial x}\right) \right) + s\left(x, t, u, \frac{\partial u}{\partial x}\right).$$

以这种形式编写的 PDE 变为

$$\pi^2 \frac{\partial u}{\partial t} = x^0 \frac{\partial}{\partial x} \left(x^0 \frac{\partial u}{\partial x} \right) + 0.$$

将方程写作适当形式后, 可知相关各项为:

$$m = 0$$

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) = \pi^2$$

$$f\left(x, t, u, \frac{\partial u}{\partial x}\right) = \frac{\partial u}{\partial x}$$

$$s\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0$$

现在, 您可以创建一个函数以编写方程代码。该函数应具有签名 **[c,f,s] = pdex1pde(x,t,u,dudx)**:

- **x** 是独立的空间变量。
- **t** 是独立的时间变量。
- **u** 是关于 **x** 和 **t** 微分的因变量。

- **dudx** 是偏空间导数 $\partial u / \partial x$ 。
- 输出 **c**、**f** 和 **s** 对应于 **pdepe** 所需的标准 PDE 形式中的系数。根据输入变量 **x**、**t**、**u** 和 **dudx** 对这些系数编写代码。

因此，此示例中的方程可以由以下函数表示：

```
function [c,f,s] = pdex1pde(x,t,u,dudx)
c = pi^2;
f = dudx;
s = 0;
end
```

(注意：所有函数都作为局部函数包含在示例的末尾。)

代码初始条件

接下来，编写一个返回初始条件的函数。初始条件应用于第一个时间值 **tspan(1)**。该函数应具有签名 **u0 = pdex1ic(x)**。

对应的函数是

```
function u0 = pdex1ic(x)
u0 = sin(pi*x);
end
```

编写边界条件代码

现在，编写计算以下边界条件的函数对于在区间 $a \leq x \leq b$ 上提出的问题，边界条件应用于所有 t 以及 $x = a$ 或 $x = b$ 。求解器所需的标准形式是

$$p(x, t, u) + q(x, t)f\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0.$$

以这种标准形式重写边界条件，并读取系数值。

对于 $x = 0$ ，方程为

$$u(0, t) = 0 \rightarrow u + 0 \cdot \frac{\partial u}{\partial x} = 0.$$

系数是：

- $p(0, t, u) = u$
- $q(0, t) = 0$

对于 $x = 1$ ，方程为

$$\pi e^{-t} + \frac{\partial u}{\partial x}(1, t) = 0 \rightarrow \pi e^{-t} + 1 \cdot \frac{\partial u}{\partial x}(1, t) = 0.$$

系数是：

- $p(1, t, u) = \pi e^{-t}$
- $q(1, t) = 1$

由于边界条件函数以 $f(x, t, u, \frac{\partial u}{\partial x})$ 形式表示，并且此项已在主 PDE 函数中定义，因此不需要在边界条件函数中指定方程的此部分。您只需在每个边界处指定 $p(x, t, u)$ 和 $q(x, t)$ 的值。

边界函数应使用函数签名 `[pl,ql,pr,qr] = pdex1bc(xl,ul,xr,ur,t)`:

- 对于左边界，输入 `xl` 和 `ul` 对应于 u 和 x 。
- 对于右边界，输入 `xr` 和 `ur` 对应于 u 和 x 。
- `t` 是独立的时间变量。
- 对于左边界，输出 `pl` 和 `ql` 对应于 $p(x, t, u)$ 和 $q(x, t)$ (对于此问题， $x = 0$)。
- 对于右边界，输出 `pr` 和 `qr` 对应于 $p(x, t, u)$ 和 $q(x, t)$ (对于此问题， $x = 1$)。

此示例中的边界条件由以下函数表示：

```
function [pl,ql,pr,qr] = pdex1bc(xl,ul,xr,ur,t)
pl = ul;
ql = 0;
pr = pi * exp(-t);
qr = 1;
end
```

选择解网格

在求解方程之前，需要指定希望用 `pdepe` 计算解的网格点 (t, x) 。将点指定为向量 `t` 和 `x`。向量 `t` 和 `x` 在求解器中的作用不同。尤其是解的成本和精确度很大程度上依赖于向量 `x` 的长度。然而，计算对向量 `t` 中的值并不敏感。

对于此问题，请使用一个网格，该网格具有 20 个位于空间区间 $[0,1]$ 中的等距点、5 个位于时间区间 $[0,2]$ 中的 `t` 值。

```
x = linspace(0,1,20);
t = linspace(0,2,5);
```

求解方程

最后，使用对称性值 `m`、PDE 方程、初始条件、边界条件以及 `x` 和 `t` 的网格来求解方程。

```
m = 0;
sol = pdepe(m,@pdex1pde,@pdex1ic,@pdex1bc,x,t);
```

`pdepe` 以三维数组 `sol` 形式返回解，其中 `sol(i,j,k)` 是在 `t(i)` 和 `x(j)` 处计算的解 u_k 的第 `k` 个分量的逼近值。`sol` 的大小是 `length(t)×length(x)×length(u0)`，因为 `u0` 为每个解分量指定初始条件。对于此问题，`u` 只有一个分量，因此 `sol` 是 $5×20$ 矩阵，但通常您可以使用命令 `u = sol(:,:,k)` 提取第 `k` 个解分量。

从 `sol` 中提取第一个解分量。

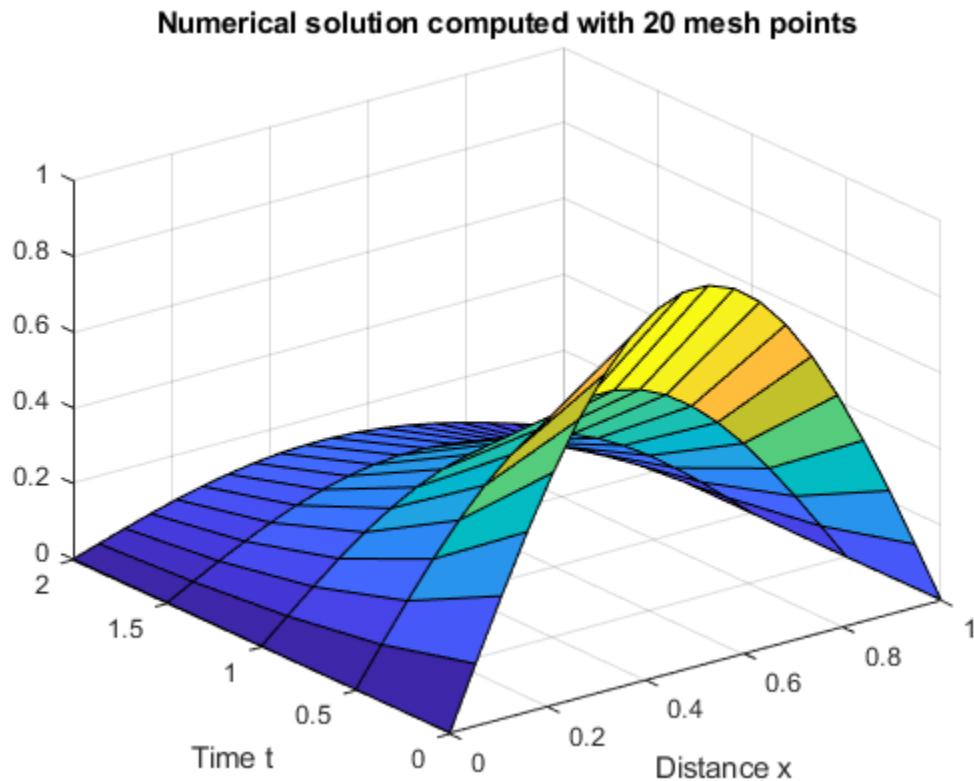
```
u = sol(:,:,1);
```

对解进行绘图

创建解的曲面图。

```
surf(x,t,u)
title('Numerical solution computed with 20 mesh points')
```

```
xlabel('Distance x')
ylabel('Time t')
```



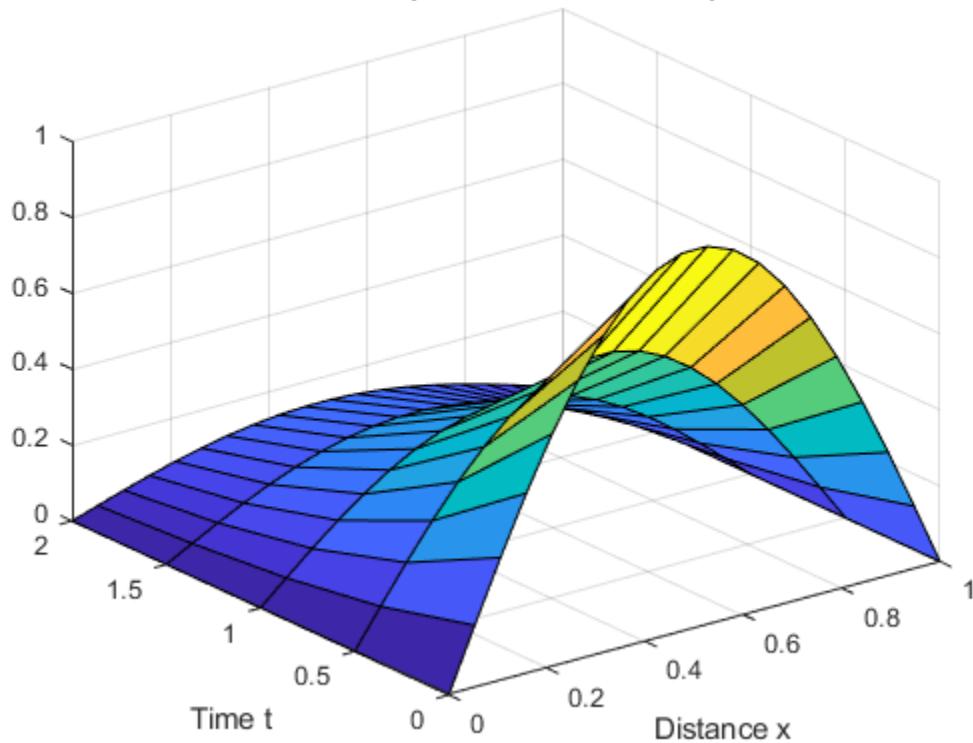
选择此问题的初始条件和边界条件，以得到解析解，由下式给出

$$u(x, t) = e^{-t} \sin(\pi x).$$

绘制采用相同网格点的解析解。

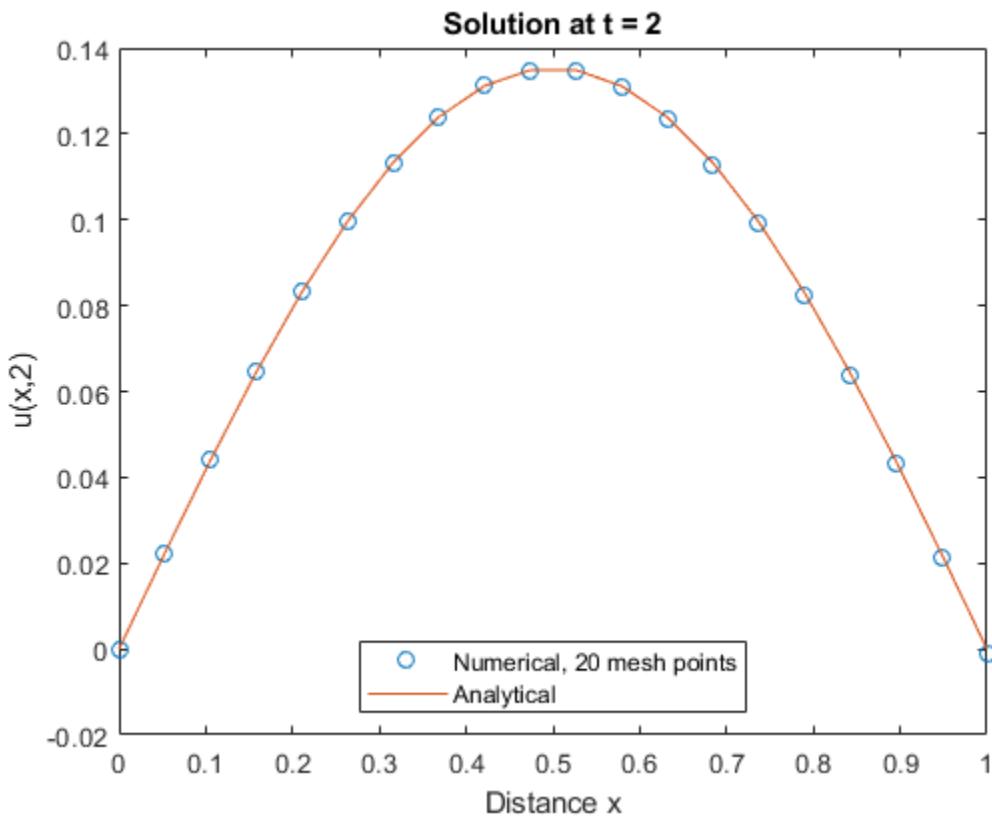
```
surf(x,t,exp(-t)*sin(pi*x))
title("True solution plotted with 20 mesh points")
xlabel('Distance x')
ylabel('Time t')
```

True solution plotted with 20 mesh points



现在，比较在 t_f (即 t 的最终值) 处的数值解和解析解。在此示例中， $t_f = 2$ 。

```
plot(x,u(end,:),'o',x,exp(-t(end))*sin(pi*x))
title('Solution at t = 2')
legend('Numerical, 20 mesh points','Analytical','Location','South')
xlabel('Distance x')
ylabel('u(x,2)')
```



局部函数

此处列出 PDE 求解器 pdepe 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```

function [c,f,s] = pdex1pde(x,t,u,dudx) % Equation to solve
c = pi^2;
f = dudx;
s = 0;
end
%
function u0 = pdex1ic(x) % Initial conditions
u0 = sin(pi*x);
end
%
function [pl,ql,pr,qr] = pdex1bc(xl,ul,xr,ur,t) % Boundary conditions
pl = ul;
ql = 0;
pr = pi * exp(-t);
qr = 1;
end
%
```

另请参阅

pdepe

详细信息

- “求解偏微分方程” (第 13-2 页)
- “求解具有不连续性的 PDE” (第 13-16 页)

求解具有不连续性的 PDE

此示例说明如何求解涉及物质界面的 PDE。物质界面使得问题在 $x = 0.5$ 处具有不连续点，初始条件在右边界 $x = 1$ 处具有不连续点。

以如下分段 PDE 为例

$$\begin{cases} \frac{\partial u}{\partial t} = x^{-2} \frac{\partial}{\partial x} \left(x^2 5 \frac{\partial u}{\partial x} \right) - 1000e^u & (0 \leq x \leq 0.5) \\ \frac{\partial u}{\partial t} = x^{-2} \frac{\partial}{\partial x} \left(x^2 \frac{\partial u}{\partial x} \right) - e^u & (0.5 \leq x \leq 1) \end{cases}$$

初始条件为

$$\begin{aligned} u(x, 0) &= 0 \quad (0 \leq x < 1), \\ u(1, 0) &= 1 \quad (x = 1). \end{aligned}$$

边界条件为

$$\begin{aligned} \frac{\partial u}{\partial x} &= 0 \quad (x = 0), \\ u(1, t) &= 1 \quad (x = 1). \end{aligned}$$

要在 MATLAB 中求解该方程，您需要对方程、初始条件和边界条件编写代码，然后在调用求解器 `pdepe` 之前选择合适的解网格。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写方程代码

在编写方程代码之前，您需要确保它的形式符合 `pdepe` 求解器的要求。`pdepe` 所需的标准形式是

$$c(x, t, u, \frac{\partial u}{\partial x}) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x} \left(x^m f(x, t, u, \frac{\partial u}{\partial x}) \right) + s(x, t, u, \frac{\partial u}{\partial x}).$$

在本例中，PDE 采用了正确形式，因此您可以读取系数的值。

$$\begin{cases} \frac{\partial u}{\partial t} = x^{-2} \frac{\partial}{\partial x} \left(x^2 5 \frac{\partial u}{\partial x} \right) - 1000e^u & (0 \leq x \leq 0.5) \\ \frac{\partial u}{\partial t} = x^{-2} \frac{\partial}{\partial x} \left(x^2 \frac{\partial u}{\partial x} \right) - e^u & (0.5 \leq x \leq 1) \end{cases}$$

通量项 $f(x, t, u, \frac{\partial u}{\partial x})$ 和源项 $s(x, t, u, \frac{\partial u}{\partial x})$ 的值根据 x 的值而变化。系数是：

$$m = 2$$

$$c(x, t, u, \frac{\partial u}{\partial x}) = 1$$

$$\begin{cases} f(x, t, u, \frac{\partial u}{\partial x}) = 5 \frac{\partial u}{\partial x} & (0 \leq x \leq 0.5) \\ f(x, t, u, \frac{\partial u}{\partial x}) = \frac{\partial u}{\partial x} & (0.5 \leq x \leq 1) \end{cases}$$

$$\begin{cases} s\left(x, t, u, \frac{\partial u}{\partial x}\right) = -1000e^u & (0 \leq x \leq 0.5) \\ s\left(x, t, u, \frac{\partial u}{\partial x}\right) = -e^u & (0.5 \leq x \leq 1) \end{cases}$$

现在，您可以创建一个函数以编写方程代码。该函数应具有签名 `[c,f,s] = pdex2pde(x,t,u,dudx)`:

- `x` 是独立的空间变量。
- `t` 是独立的时间变量。
- `u` 是关于 `x` 和 `t` 微分的因变量。
- `dudx` 是偏空间导数 $\partial u / \partial x$ 。
- 输出 `c`、`f` 和 `s` 对应于 `pdepe` 所需的标准 PDE 形式中的系数。根据输入变量 `x`、`t`、`u` 和 `dudx` 对这些系数编写代码。

因此，此示例中的方程可以由以下函数表示：

```
function [c,f,s] = pdex2pde(x,t,u,dudx)
c = 1;
if x <= 0.5
    f = 5*dudx;
    s = -1000*exp(u);
else
    f = dudx;
    s = -exp(u);
end
end
```

(注意：所有函数都作为局部函数包含在示例的末尾。)

编写初始条件代码

接下来，编写一个返回初始条件的函数。初始条件应用在第一个时间值处，并为 `x` 的任何值提供 $u(x, t_0)$ 的值。使用函数签名 `u0 = pdex2ic(x)` 编写函数。

初始条件为

$$\begin{aligned} u(x, 0) &= 0 & (0 \leq x < 1), \\ u(1, 0) &= 1 & (x = 1). \end{aligned}$$

对应的函数是

```
function u0 = pdex2ic(x)
if x < 1
    u0 = 0;
else
    u0 = 1;
end
end
```

编写边界条件代码

现在，编写一个计算边界条件的函数。对于区间 $a \leq x \leq b$ 上的问题，边界条件应用于所有 `t` 以及 $x = a$ 或 $x = b$ 的情形。求解器所需的边界条件的标准形式是

$$p(x, t, u) + q(x, t)f\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0.$$

由于此示例具有球面对称性 ($m = 2$)，**pdepe** 求解器会自动强制执行左边界条件以约束在原点的解，并忽略在边界函数中为左边界指定的任何条件。因此，对于左边界条件，您可以指定 $p_L = q_L = 0$ 。对于右边界条件，您可以用标准形式重写边界条件，并读取 p_R 和 q_R 的系数值。

对于 $x = 1$ ，方程为 $u(1, t) = 1 \rightarrow (u - 1) + 0 \cdot \frac{\partial u}{\partial x} = 0$ 。系数是：

- $p_R(1, t, u) = u - 1$
- $q_R(1, t) = 0$

边界函数应使用函数签名 `[pl,ql,pr,qr] = pdex2bc(xl,ul,xr,ur,t)`：

- 对于左边界，输入 `xl` 和 `ul` 对应于 `u` 和 `x`。
- 对于右边界，输入 `xr` 和 `ur` 对应于 `u` 和 `x`。
- `t` 是独立的时间变量。
- 对于左边界，输出 `pl` 和 `ql` 对应于 $p_L(x, t, u)$ 和 $q_L(x, t)$ （对于此问题， $x = 0$ ）。
- 对于右边界，输出 `pr` 和 `qr` 对应于 $p_R(x, t, u)$ 和 $q_R(x, t)$ （对于此问题， $x = 1$ ）。

此示例中的边界条件由以下函数表示：

```
function [pl,ql,pr,qr] = pdex2bc(xl,ul,xr,ur,t)
pl = 0;
ql = 0;
pr = ur - 1;
qr = 0;
end
```

选择解网格

空间网格应包括 $x = 0.5$ 附近的几个值以表示不连续界面，并包括 $x = 1$ 附近的点，因为在该点上具有不一致的初始值 ($u(1, 0) = 1$) 和边界值 ($u(1, t) = 0$)。对于较小的 t ，解的变化很快，因此请使用可以解析这种急剧变化的时间步。

```
x = [0 0.1 0.2 0.3 0.4 0.45 0.475 0.5 0.525 0.55 0.6 0.7 0.8 0.9 0.95 0.975 0.99 1];
t = [0 0.001 0.005 0.01 0.05 0.1 0.5 1];
```

求解方程

最后，使用对称性值 `m`、PDE 方程、初始条件、边界条件以及 `x` 和 `t` 的网格来求解方程。

```
m = 2;
sol = pdepe(m,@pdex2pde,@pdex2ic,@pdex2bc,x,t);
```

pdepe 以三维数组 `sol` 形式返回解，其中 `sol(i,j,k)` 是在 `t(i)` 和 `x(j)` 处计算的解 u_k 的第 `k` 个分量的逼近值。`sol` 的大小是 `length(t)×length(x)×length(u0)`，因为 `u0` 为每个解分量指定初始条件。对于此问题，`u` 只有一个分量，因此 `sol` 是 8×18 矩阵，但通常您可以使用命令 `u = sol(:,:,k)` 提取第 `k` 个解分量。

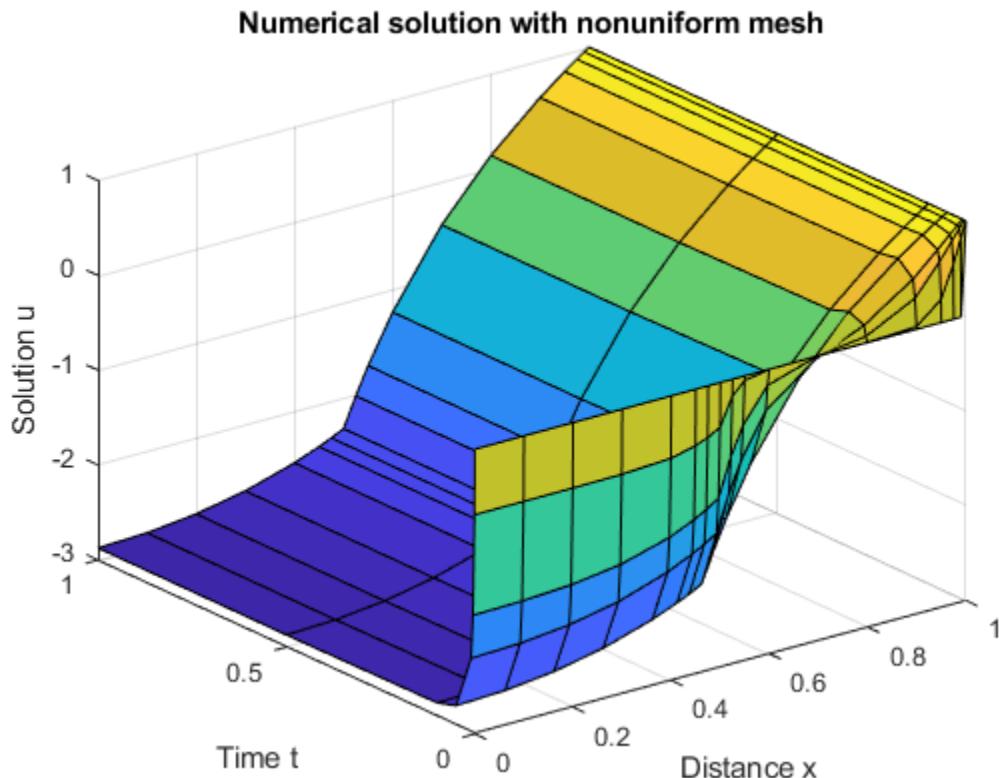
从 `sol` 中提取第一个解分量。

```
u = sol(:,:,1);
```

对解进行绘图

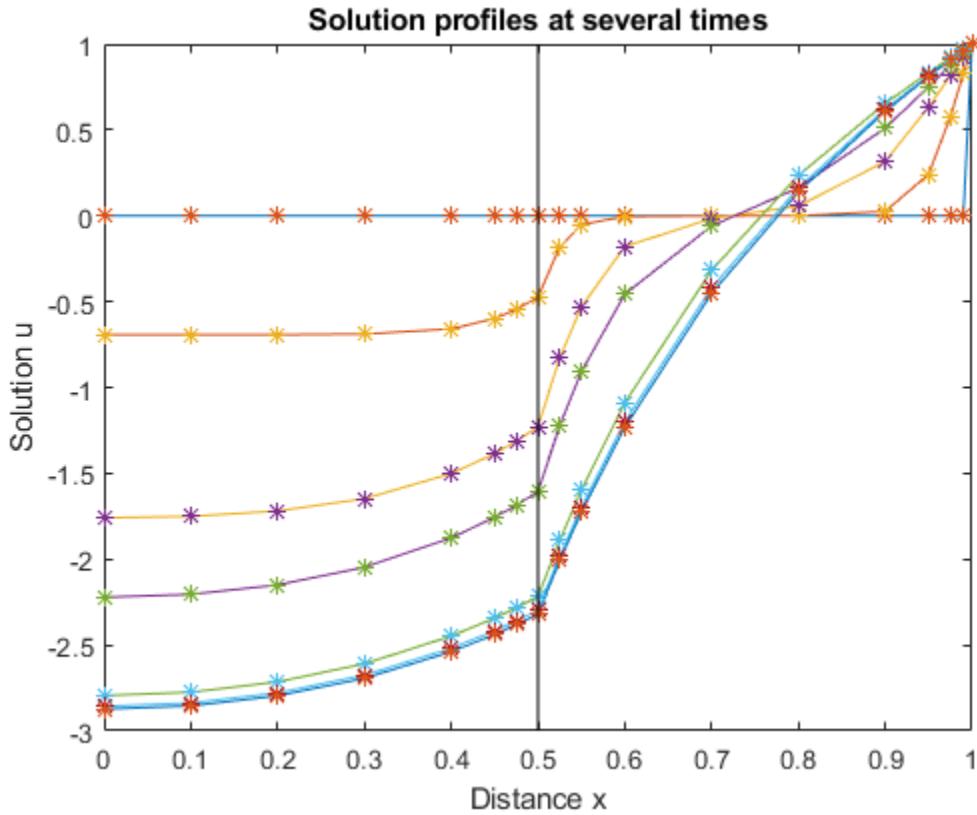
创建在 x 和 t 的所选网格点上绘制的解 u 的曲面图。由于 $m = 2$ 问题是在具有球面对称性的球面几何中提出的，因此解仅在径向 x 方向上变化。

```
surf(x,t,u)
title('Numerical solution with nonuniform mesh')
xlabel('Distance x')
ylabel('Time t')
zlabel('Solution u')
```



现在，只需绘制 x 和 u 即可获得曲面图中等高线的侧视图。在 $x = 0.5$ 处添加一条线，以突出材料接口的效果。

```
plot(x,u,x,u,'*')
line([0.5 0.5], [-3 1], 'Color', 'k')
xlabel('Distance x')
ylabel('Solution u')
title('Solution profiles at several times')
```



局部函数

此处列出 PDE 求解器 `pdepe` 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```

function [c,f,s] = pdex2pde(x,t,u,dudx) % Equation to solve
c = 1;
if x <= 0.5
    f = 5*dudx;
    s = -1000*exp(u);
else
    f = dudx;
    s = -exp(u);
end
end
%-----
function u0 = pdex2ic(x) %Initial conditions
if x < 1
    u0 = 0;
else
    u0 = 1;
end
end
%
function [pl,ql,pr,qr] = pdex2bc(xl,ul,xr,ur,t) % Boundary conditions
pl = 0;
ql = 0;

```

```
pr = ur - 1;  
qr = 0;  
end  
%
```

另请参阅

[pdepe](#)

详细信息

- “求解偏微分方程” (第 13-2 页)
- “求解 PDE 并计算偏导数” (第 13-22 页)

求解 PDE 并计算偏导数

此示例说明如何求解一个晶体管偏微分方程 (PDE)，并使用结果获得偏导数，这是求解更大型问题的一部分。

以如下 PDE 为例

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2} - \frac{D\eta}{L} \frac{\partial u}{\partial x}.$$

此方程出现在晶体管理论 [1] 中， $u(x, t)$ 是描述 PNP 晶体管基极中过剩电荷载流子（或空穴）浓度的函数。 D 和 η 是物理常量。该公式在区间 $0 \leq x \leq L$ 上对于时间 $t \geq 0$ 成立。

初始条件包括常量 K ，由下式给出

$$u(x, 0) = \frac{KL}{D} \left(\frac{1 - e^{-\eta(1-x/L)}}{\eta} \right).$$

该问题具有由下式给出的边界条件

$$u(0, t) = u(L, t) = 0.$$

对于固定 x ，方程 $u(x, t)$ 的解将过剩电荷的坍塌描述为 $t \rightarrow \infty$ 。这种坍塌产生一种电流，称为发射极放电电流，它还有另一个常量 I_p ：

$$I(t) = \left[\frac{I_p D}{K} \frac{\partial}{\partial x} u(x, t) \right]_{x=0}.$$

由于在 $t = 0$ 和 $t > 0$ 时， $x = 0$ 处的边界值不一致，该公式对 $t > 0$ 有效。由于 PDE 对 $u(x, t)$ 有闭型级数解，您可以通过解析方式和数值方式计算发射极放电电流，并对结果进行比较。

要在 MATLAB 中求解此问题，您需要对 PDE、方程、初始条件和边界条件编写代码，然后在调用求解器 `pdepe` 之前选择合适的解网格。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

定义物理常量

要跟踪物理常量，请创建一个结构体数组，其中每个常量都有一个对应的字段。当您稍后为方程、初始条件和边界条件定义函数时，可以将此结构体作为额外的参数传入，以便函数可以访问常量。

```
C.L = 1;
C.D = 0.1;
C.eta = 10;
C.K = 1;
C.Ip = 1;
```

编写方程代码

在编写方程代码之前，您需要确保它的形式符合 `pdepe` 求解器的要求：

$$c(x, t, u, \frac{\partial u}{\partial t}) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x} \left(x^m f(x, t, u, \frac{\partial u}{\partial x}) \right) + s(x, t, u, \frac{\partial u}{\partial x}).$$

此形式的 PDE 为

$$\frac{\partial u}{\partial t} = x^0 \frac{\partial}{\partial x} \left(x^0 D \frac{\partial u}{\partial x} \right) - \frac{D\eta}{L} \frac{\partial u}{\partial x}.$$

因此，方程中的系数的值是

- $m = 0$ (没有角对称性的笛卡尔坐标)
- $c(x, t, u, \frac{\partial u}{\partial x}) = 1$
- $f(x, t, u, \frac{\partial u}{\partial x}) = D \frac{\partial u}{\partial x}$
- $s(x, t, u, \frac{\partial u}{\partial x}) = - \frac{D\eta}{L} \frac{\partial u}{\partial x}$

现在，您可以创建一个函数以编写方程代码。该函数应具有签名 `[c,f,s] = transistorPDE(x,t,u,dudx,C)`:

- **x** 是独立的空间变量。
- **t** 是独立的时间变量。
- **u** 是关于 **x** 和 **t** 微分的因变量。
- **dudx** 是偏空间导数 $\partial u / \partial x$ 。
- **C** 是包含物理常量的额外输入。
- 输出 **c**、**f** 和 **s** 对应于 `pdepe` 所需的标准 PDE 形式中的系数。

因此，此示例中的方程可以由以下函数表示：

```
function [c,f,s] = transistorPDE(x,t,u,dudx,C)
D = C.D;
eta = C.eta;
L = C.L;

c = 1;
f = D*dudx;
s = -(D*eta/L)*dudx;
end
```

(注意：所有函数都作为局部函数包含在示例的末尾。)

代码初始条件

接下来，编写一个返回初始条件的函数。初始条件应用在第一个时间值处，并为 **x** 的任何值提供 $u(x, t_0)$ 的值。使用函数签名 `u0 = transistorIC(x,C)` 编写函数。

初始条件为

$$u(x, 0) = \frac{KL}{D} \left(\frac{1 - e^{-\eta(1 - x/L)}}{\eta} \right).$$

对应的函数是

```
function u0 = transistorIC(x,C)
K = C.K;
L = C.L;
D = C.D;
```

```

eta = C.eta;
u0 = (K*L/D)*(1 - exp(-eta*(1 - x/L)))/eta;
end

```

编写边界条件代码

现在，编写一个计算边界条件 $u(0, t) = u(1, t) = 0$ 的函数。对于在区间 $a \leq x \leq b$ 上提出的问题，边界条件应用于所有 t 以及 $x = a$ 或 $x = b$ 。求解器所需的边界条件的标准形式是

$$p(x, t, u) + q(x, t) f\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0.$$

以这种形式编写的此问题的边界条件是

- 对于 $x = 0$ ，方程为 $u + 0 \cdot d\frac{\partial u}{\partial x} = 0$. 系数为：

- $p_L(x, t, u) = u$,
- $q_L(x, t) = 0$.

- 同样，对于 $x = 1$ ，方程为 $u + 0 \cdot d\frac{\partial u}{\partial x} = 0$. 系数为：

- $p_R(x, t, u) = u$,
- $q_R(x, t) = 0$.

边界函数应使用函数签名 `[pl,ql,pr,qr] = transistorBC(xl,ul,xr,ur,t)`:

- 对于左边界，输入 `xl` 和 `ul` 对应于 x 和 u 。
- 对于右边界，输入 `xr` 和 `ur` 对应于 x 和 u 。
- `t` 是独立的时间变量。
- 对于左边界，输出 `pl` 和 `ql` 对应于 $p_L(x, t, u)$ 和 $q_L(x, t)$ (对于此问题， $x = 0$)。
- 对于右边界，输出 `pr` 和 `qr` 对应于 $p_R(x, t, u)$ 和 $q_R(x, t)$ (对于此问题， $x = 1$)。

此示例中的边界条件由以下函数表示：

```

function [pl,ql,pr,qr] = transistorBC(xl,ul,xr,ur,t)
pl = ul;
ql = 0;
pr = ur;
qr = 0;
end

```

选择解网格

解网格定义 x 和 t 的值，求解器基于它们来计算解。由于此问题的解变化很快，请使用一个相对精细的网格，其中包含 50 个位于 $0 \leq x \leq L$ 区间中的空间点和 50 个位于 $0 \leq t \leq 1$ 区间中的时间点。

```

x = linspace(0,C.L,50);
t = linspace(0,1,50);

```

求解方程

最后，使用对称性值 m 、PDE 方程、初始条件、边界条件以及 x 和 t 的网格来求解方程。由于 `pdepe` 需要 PDE 函数使用四个输入、初始条件函数使用一个输入，请创建函数句柄，将由物理常量组成的结构体作为额外输入来传入。

```
m = 0;
eqn = @(x,t,u,dudx) transistorPDE(x,t,u,dudx,C);
ic = @(x) transistorIC(x,C);
sol = pdepe(m,eqn,ic,@transistorBC,x,t);
```

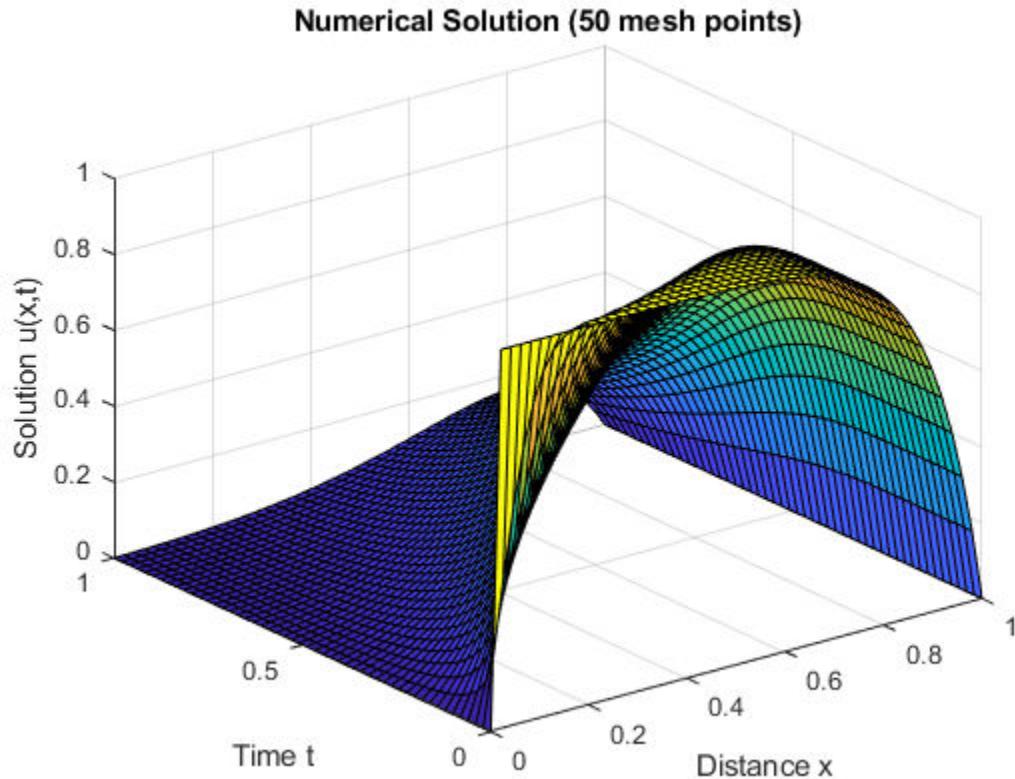
`pdepe` 以三维数组 `sol` 形式返回解，其中 `sol(i,j,k)` 是在 $t(i)$ 和 $x(j)$ 处计算的解 u_k 的第 k 个分量的逼近值。对于此问题，`u` 只有一个分量，但通常您可以使用命令 `u = sol(:,:,k)` 提取第 k 个解分量。

```
u = sol(:,:,1);
```

对解进行绘图

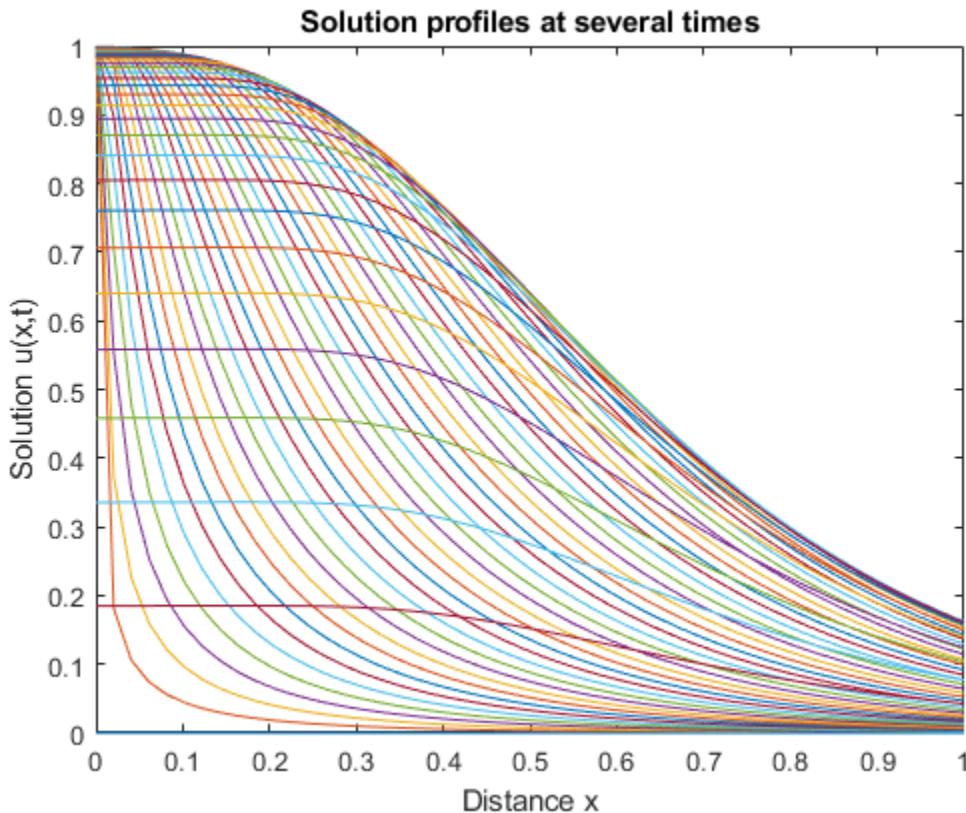
创建在 x 和 t 的所选网格点上绘制的解 u 的曲面图。

```
surf(x,t,u)
title('Numerical Solution (50 mesh points)')
xlabel('Distance x')
ylabel('Time t')
zlabel('Solution u(x,t)')
```



现在，只需绘制 x 和 u 即可获得曲面图中等高线的侧视图。

```
plot(x,u)
xlabel('Distance x')
ylabel('Solution u(x,t)')
title('Solution profiles at several times')
```



计算发射极放电电流

使用 $u(x, t)$ 的级数解，发射极放电电流可以表示为无穷级数 [1]:

$$I(t) = 2\pi^2 I_p \left(\frac{1 - e^{-\eta}}{\eta} \right) \sum_{n=1}^{\infty} \frac{n^2}{n^2 \pi^2 + \eta^2/4} e^{-\frac{dt}{L^2} (n^2 \pi^2 + \eta^2/4)}.$$

编写一个函数，以使用级数中的 40 个项计算 $I(t)$ 的解析解。唯一的变量是时间，但要将常量结构体指定为函数的另一个输入。

```
function It = serex3(t,C) % Approximate I(t) by series expansion.
Ip = C.Ip;
eta = C.eta;
D = C.D;
L = C.L;

It = 0;
for n = 1:40 % Use 40 terms
    m = (n*pi)^2 + 0.25*eta^2;
    It = It + ((n*pi)^2 / m)* exp(-(D/L^2)*m*t);
end
```

```
It = 2*Ip*((1 - exp(-eta))/eta)*It;
end
```

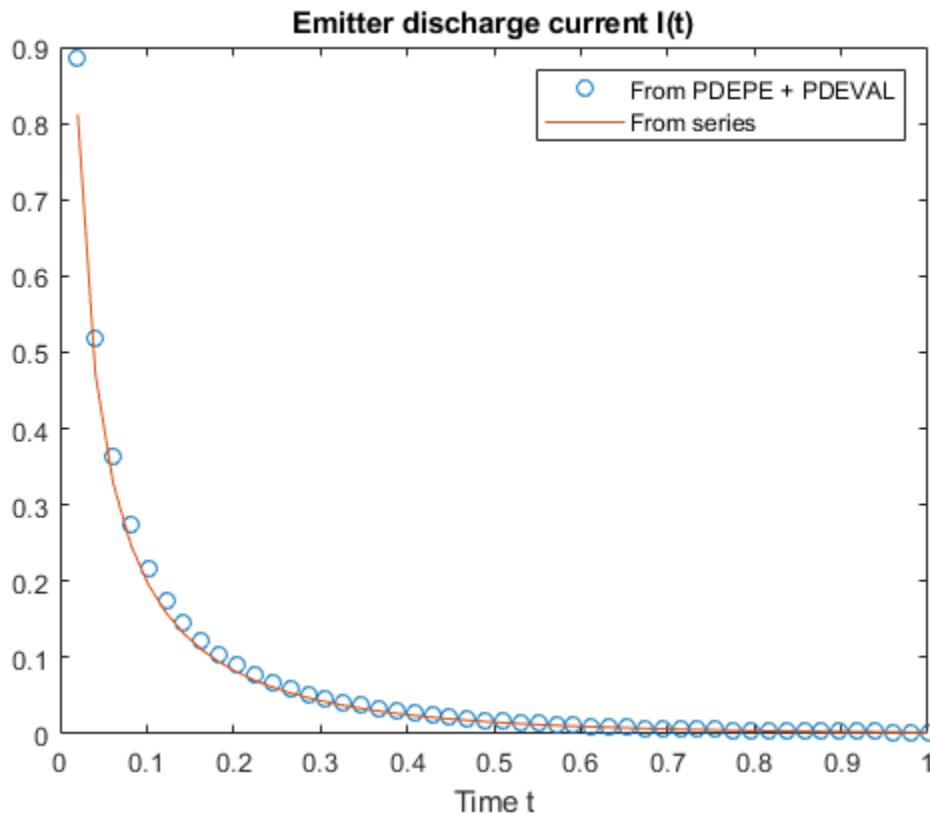
使用 **pdepe** 计算的 $u(x, t)$ 的数值解，您还可以通过以下方程计算在 $x = 0$ 处的 $I(t)$ 的数值逼近

$$I(t) = \left[\frac{I_p D}{K} \frac{\partial}{\partial x} u(x, t) \right]_{x=0}$$

计算 $I(t)$ 的解析解和数值解，并对结果绘图。使用 **pdeval** 计算 $\partial u / \partial x$ 在 $x = 0$ 处的值。

```
nt = length(t);
I = zeros(1,nt);
seriesI = zeros(1,nt);
iok = 2:nt;
for j = iok
    % At time t(j), compute du/dx at x = 0.
    [~,I(j)] = pdeval(m,x,u(j,:),0);
    seriesI(j) = serex3(t(j),C);
end
% Numeric solution has form I(t) = (I_p*D/K)*du(0,t)/dx
I = (C.Ip*C.D/C.K)*I;

plot(t(iok),I(iok),'o',t(iok),seriesI(iok))
legend('From PDEPE + PDEVAL','From series')
title('Emitter discharge current I(t)')
xlabel('Time t')
```



结果相当吻合。通过使用更精细的解网格，您可以进一步改进 **pdepe** 得出的数值结果。

局部函数

此处列出 PDE 求解器 **pdepe** 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```

function [c,f,s] = transistorPDE(x,t,u,dudx,C) % Equation to solve
D = C.D;
eta = C.eta;
L = C.L;

c = 1;
f = D*dudx;
s = -(D*eta/L)*dudx;
end
% -----
function u0 = transistorIC(x,C) % Initial condition
K = C.K;
L = C.L;
D = C.D;
eta = C.eta;

u0 = (K*L/D)*(1 - exp(-eta*(1 - x/L)))/eta;
end
% -----
function [pl,ql,pr,qr] = transistorBC(xl,ul,xr,ur,t) % Boundary conditions
pl = ul;
ql = 0;
pr = ur;
qr = 0;
end
% -----
function It = serex3(t,C) % Approximate I(t) by series expansion.
Ip = C.Ip;
eta = C.eta;
D = C.D;
L = C.L;

It = 0;
for n = 1:40 % Use 40 terms
    m = (n*pi)^2 + 0.25*eta^2;
    It = It + ((n*pi)^2 / m)* exp(-(D/L^2)*m*t);
end
It = 2*Ip*((1 - exp(-eta))/eta)*It;
end
%
```

参考

[1] Zachmanoglou, E.C. and D.L. Thoe. Introduction to Partial Differential Equations with Applications. Dover, New York, 1986.

另请参阅

pdepe

详细信息

- “求解偏微分方程” (第 13-2 页)
- “求解 PDE 方程组” (第 13-30 页)

求解 PDE 方程组

此示例说明由两个偏微分方程构成的方程组的解的构成，以及如何对解进行计算和绘图。

以如下 PDE 方程组为例

$$\frac{\partial u_1}{\partial t} = 0.024 \frac{\partial^2 u_1}{\partial x^2} - F(u_1 - u_2),$$

$$\frac{\partial u_2}{\partial t} = 0.170 \frac{\partial^2 u_2}{\partial x^2} + F(u_1 - u_2).$$

(函数 $F(y) = e^{5.73y} - e^{-11.46y}$ 用作速记形式。)

该公式在区间 $0 \leq x \leq 1$ 上对于时间 $t \geq 0$ 成立。初始条件为

$$u_1(x, 0) = 1,$$

$$u_2(x, 0) = 0.$$

边界条件为

$$\frac{\partial}{\partial x} u_1(0, t) = 0,$$

$$u_2(0, t) = 0,$$

$$\frac{\partial}{\partial x} u_2(1, t) = 0,$$

$$u_1(1, t) = 1.$$

要在 MATLAB 中求解该方程，您需要对方程、初始条件和边界条件编写代码，然后在调用求解器 **pdepe** 之前选择合适的解网格。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写方程代码

在编写方程代码之前，您需要确保它的形式符合 **pdepe** 求解器的要求：

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x} \left(x^m f\left(x, t, u, \frac{\partial u}{\partial x}\right) \right) + s\left(x, t, u, \frac{\partial u}{\partial x}\right).$$

在此形式中，PDE 系数是矩阵值，方程变为

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \frac{\partial}{\partial t} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \frac{\partial}{\partial x} \begin{bmatrix} 0.024 \frac{\partial u_1}{\partial x} \\ 0.170 \frac{\partial u_2}{\partial x} \end{bmatrix} + \begin{bmatrix} -F(u_1 - u_2) \\ F(u_1 - u_2) \end{bmatrix}.$$

因此，方程中的系数的值是

$$m = 0$$

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad (\text{仅对角线值})$$

$$f\left(x, t, u, \frac{\partial u}{\partial x}\right) = \begin{bmatrix} 0.024 \frac{\partial u_1}{\partial x} \\ 0.170 \frac{\partial u_2}{\partial x} \end{bmatrix}$$

$$s\left(x, t, u, \frac{\partial u}{\partial x}\right) = \begin{bmatrix} -F(u_1 - u_2) \\ F(u_1 - u_2) \end{bmatrix}$$

现在，您可以创建一个函数以编写方程代码。该函数应具有签名 `[c,f,s] = pdefun(x,t,u,dudx)`：

- **x** 是独立的空间变量。
- **t** 是独立的时间变量。
- **u** 是关于 **x** 和 **t** 微分的因变量。它是二元素向量，其中 **u(1)** 是 $u_1(x, t)$ ，**u(2)** 是 $u_2(x, t)$ 。
- **dudx** 是偏空间导数 $\partial u / \partial x$ 。它是二元素向量，其中 **dudx(1)** 是 $\partial u_1 / \partial x$ ，**dudx(2)** 是 $\partial u_2 / \partial x$ 。
- 输出 **c**、**f** 和 **s** 对应于 **pdepe** 所需的标准 PDE 形式中的系数。

因此，此示例中的方程可由以下函数表示：

```
function [c,f,s] = pdefun(x,t,u,dudx)
c = [1; 1];
f = [0.024; 0.17] .* dudx;
y = u(1) - u(2);
F = exp(5.73*y)-exp(-11.47*y);
s = [-F; F];
end
```

(注意：所有函数都作为局部函数包含在示例的末尾。)

编写初始条件代码

接下来，编写一个返回初始条件的函数。初始条件应用在第一个时间值处，并为 **x** 的任何值提供 $u(x, t_0)$ 的值。初始条件的数量必须等于方程的数量，因此对于此问题，有两个初始条件。使用函数签名 **u0 = pdeic(x)** 编写函数。

初始条件为

$$u_1(x, 0) = 1,$$

$$u_2(x, 0) = 0.$$

对应的函数是

```
function u0 = pdeic(x)
u0 = [1; 0];
end
```

编写边界条件代码

现在，编写计算以下边界条件的函数

$$\frac{\partial}{\partial x} u_1(0, t) = 0,$$

$$u_2(0, t) = 0,$$

$$\frac{\partial}{\partial x} u_2(1, t) = 0,$$

$$u_1(1, t) = 1.$$

对于在区间 $a \leq x \leq b$ 上提出的问题，边界条件应用于所有 t 以及 $x = a$ 或 $x = b$ 。求解器所需的标准形式是

$$p(x, t, u) + q(x, t) f\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0.$$

以这种形式编写， u 的偏导数的边界条件需要用通量 $f\left(x, t, u, \frac{\partial u}{\partial x}\right)$ 来表示。因此，此问题的边界条件是

对于 $x = 0$ ，方程为

$$\begin{bmatrix} 0 \\ u_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0.024 \frac{\partial u_1}{\partial x} \\ 0.170 \frac{\partial u_2}{\partial x} \end{bmatrix} = 0.$$

系数是：

$$p_L(x, t, u) = \begin{bmatrix} 0 \\ u_2 \end{bmatrix},$$

$$q_L(x, t) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

同样，对于 $x = 1$ ，方程是

$$\begin{bmatrix} u_1 - 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 0.024 \frac{\partial u_1}{\partial x} \\ 0.170 \frac{\partial u_2}{\partial x} \end{bmatrix} = 0.$$

系数是：

$$p_R(x, t, u) = \begin{bmatrix} u_1 - 1 \\ 0 \end{bmatrix},$$

$$q_R(x, t) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

边界函数应使用函数签名 `[pl,ql,pr,qr] = pdebc(xl,ul,xr,ur,t)`：

- 对于左边界，输入 `xl` 和 `ul` 对应于 u 和 x 。
- 对于右边界，输入 `xr` 和 `ur` 对应于 u 和 x 。
- t 是独立的时间变量。
- 对于左边界，输出 `pl` 和 `ql` 对应于 $p_L(x, t, u)$ 和 $q_L(x, t)$ （对于此问题， $x = 0$ ）。
- 对于右边界，输出 `pr` 和 `qr` 对应于 $p_R(x, t, u)$ 和 $q_R(x, t)$ （对于此问题， $x = 1$ ）。

此示例中的边界条件由以下函数表示：

```
function [pl,ql,pr,qr] = pdebc(xl,ul,xr,ur,t)
pl = [0; ul(2)];
ql = [1; 0];
pr = [ur(1)-1; 0];
qr = [0; 1];
end
```

选择解网格

当 t 较小时，此问题的解会快速变化。虽然 **pdepe** 选择了适合解析急剧变化的时间步，但要在输出绘图中显示该行为，您需要选择适当的输出时间。对于空间网格，在 $0 \leq x \leq 1$ 两端的解中都存在边界层，因此您需要在那里指定网格点来解析急剧变化。

```
x = [0 0.005 0.01 0.05 0.1 0.2 0.5 0.7 0.9 0.95 0.99 0.995 1];
t = [0 0.005 0.01 0.05 0.1 0.5 1 1.5 2];
```

求解方程

最后，使用对称性值 m 、PDE 方程、初始条件、边界条件以及 x 和 t 的网格来求解方程。

```
m = 0;
sol = pdepe(m,@pdefun,@pdeic,@pdebc,x,t);
```

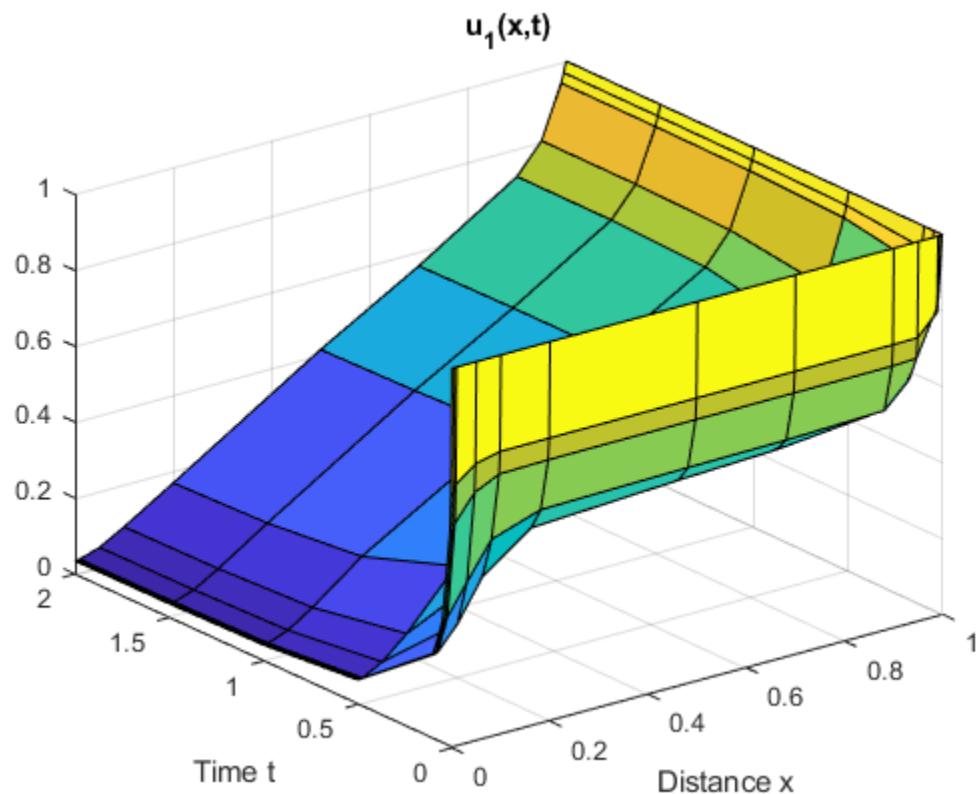
pdepe 以三维数组 **sol** 形式返回解，其中 **sol(i,j,k)** 是在 **t(i)** 和 **x(j)** 处计算的解 u_k 的第 **k** 个分量的逼近值。将每个解分量提取到一个单独变量中。

```
u1 = sol(:,:,1);
u2 = sol(:,:,2);
```

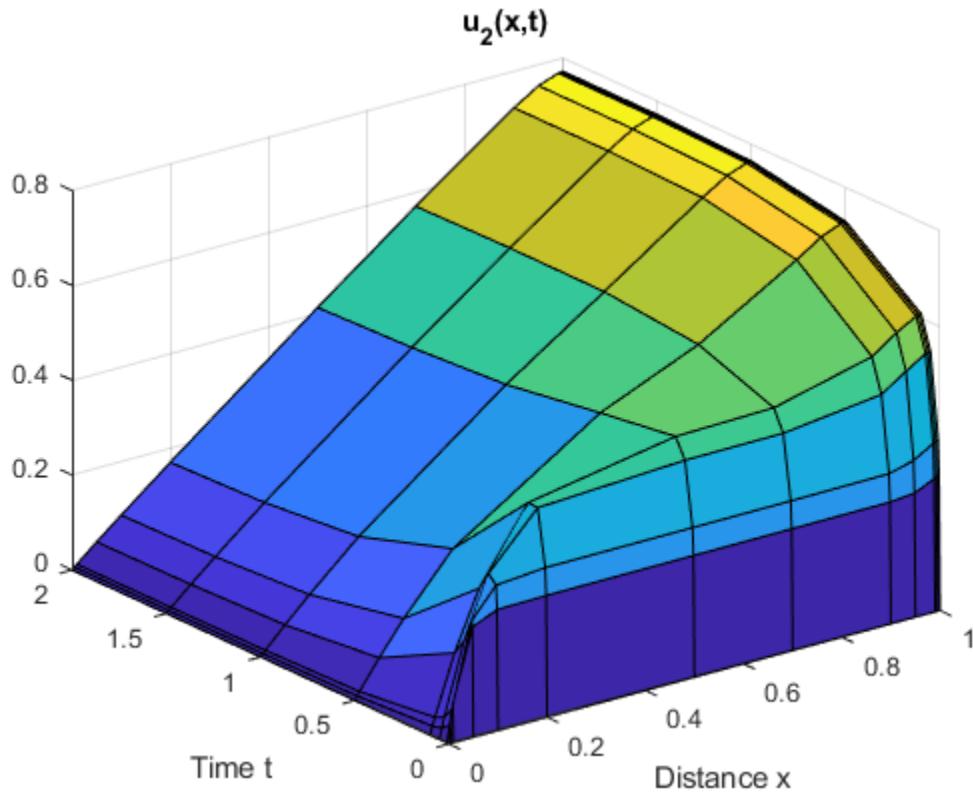
对解进行绘图

创建在 x 和 t 的所选网格点上绘制的 u_1 和 u_2 的解的曲面图。

```
surf(x,t,u1)
title('u_1(x,t)')
xlabel('Distance x')
ylabel('Time t')
```



```
surf(x,t,u2)
title('u_2(x,t)')
xlabel('Distance x')
ylabel('Time t')
```



局部函数

此处列出 PDE 求解器 `pdepe` 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```

function [c,f,s] = pdefun(x,t,u,dudx) % Equation to solve
c = [1; 1];
f = [0.024; 0.17] .* dudx;
y = u(1) - u(2);
F = exp(5.73*y)-exp(-11.47*y);
s = [-F; F];
end
%
function u0 = pdeic(x) % Initial Conditions
u0 = [1; 0];
end
%
function [pl,ql,pr,qr] = pdebc(xl,ul,xr,ur,t) % Boundary Conditions
pl = [0; ul(2)];
ql = [1; 0];
pr = [ur(1)-1; 0];
qr = [0; 1];
end
%

```

另请参阅

`pdepe`

详细信息

- “求解偏微分方程” (第 13-2 页)
- “求解单个 PDE” (第 13-9 页)
- “使用初始条件阶跃函数求解 PDE 方程组” (第 13-37 页)

使用初始条件阶跃函数求解 PDE 方程组

此示例说明如何求解初始条件中使用步函数的偏微分方程组。

以如下 PDE 为例

$$\begin{aligned}\frac{\partial n}{\partial t} &= \left(d \frac{\partial^2 n}{\partial x^2} - a n \frac{\partial^2 c}{\partial x^2} \right) + S r n(N - n), \\ \frac{\partial c}{\partial t} &= \frac{\partial^2 c}{\partial x^2} + S \left(\frac{n}{n+1} - c \right).\end{aligned}$$

这些方程涉及常量参数 d 、 a 、 S 、 r 和 N ，并为 $0 \leq x \leq 1$ 和 $t \geq 0$ 定义。这些方程源于描述肿瘤相关血管生成的初始步骤的数学模型 [1]。 $n(x, t)$ 表示内皮细胞的细胞密度， $c(x, t)$ 表示因为肿瘤而释放的蛋白质的浓度。

当出现以下情况时，此问题实现常数稳态

$$\begin{bmatrix} n_0 \\ c_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}.$$

然而，稳定性分析预测方程组会演化出非齐次解 [1]。因此，需要使用阶跃函数作为初始条件，以扰动稳态和促进方程组演化。

边界条件要求两个解分量在 $x = 0$ 和 $x = 1$ 处通量为零。

$$\begin{aligned}\frac{\partial}{\partial x} n(0, t) &= \frac{\partial}{\partial x} n(1, t) = 0, \\ \frac{\partial}{\partial x} c(0, t) &= \frac{\partial}{\partial x} c(1, t) = 0.\end{aligned}$$

要在 MATLAB 中求解此方程组，您需要对方程、初始条件和边界条件编写代码，然后在调用求解器 **pdepe** 之前选择合适的解网格。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写方程代码

在编写方程代码之前，您需要确保它的形式符合 **pdepe** 求解器的要求：

$$c(x, t, u, \frac{\partial u}{\partial x}) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x} \left(x^m f(x, t, u, \frac{\partial u}{\partial x}) \right) + s(x, t, u, \frac{\partial u}{\partial x}).$$

由于该 PDE 方程组中有两个方程，PDE 方程组可以重写为

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \frac{\partial}{\partial t} \begin{bmatrix} n \\ c \end{bmatrix} = \frac{\partial}{\partial x} \begin{bmatrix} d \frac{\partial n}{\partial x} - a n \frac{\partial c}{\partial x} \\ \frac{\partial c}{\partial x} \end{bmatrix} + \begin{bmatrix} S r n(N - n) \\ S \left(\frac{n}{n+1} - c \right) \end{bmatrix}.$$

则方程中系数的值为

$$m = 0$$

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ (仅对角线值)}$$

$$f\left(x, t, u, \frac{\partial u}{\partial x}\right) = \begin{bmatrix} d \frac{\partial n}{\partial x} - a n \frac{\partial c}{\partial x} \\ \frac{\partial c}{\partial x} \end{bmatrix}$$

$$s\left(x, t, u, \frac{\partial u}{\partial x}\right) = \begin{bmatrix} S r n(N - n) \\ S \left(\frac{n}{n + 1} - c\right) \end{bmatrix}$$

现在，您可以创建一个函数以编写方程代码。该函数应具有签名 `[c,f,s] = angiopde(x,t,u,dudx)`:

- `x` 是独立的空间变量。
- `t` 是独立的时间变量。
- `u` 是关于 `x` 和 `t` 微分的因变量。它是二元素向量，其中 `u(1)` 是 $n(x, t)$, `u(2)` 是 $c(x, t)$ 。
- `dudx` 是偏空间导数 $\partial u / \partial x$ 。它是二元素向量，其中 `dudx(1)` 是 $\partial n / \partial x$, `dudx(2)` 是 $\partial c / \partial x$ 。
- 输出 `c`、`f` 和 `s` 对应于 `pdepe` 所需的标准 PDE 形式中的系数。

因此，此示例中的方程可由以下函数表示：

```
function [c,f,s] = angiopde(x,t,u,dudx)
d = 1e-3;
a = 3.8;
S = 3;
r = 0.88;
N = 1;

c = [1; 1];
f = [d*dudx(1) - a*u(1)*dudx(2)
      dudx(2)];
s = [S*r*u(1)*(N - u(1));
      S*(u(1)/(u(1) + 1) - u(2))];
end
```

(注意：所有函数都作为局部函数包含在示例的末尾。)

编写初始条件代码

接下来，编写一个返回初始条件的函数。初始条件应用在第一个时间值处，并为 `x` 的任何值提供 $n(x, t_0)$ 和 $c(x, t_0)$ 的值。使用函数签名 `u0 = angioic(x)` 编写函数。

当出现以下情况时，此问题实现常数稳态

$$\begin{bmatrix} n_0 \\ c_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}.$$

然而，稳定性分析预测方程组会演化出非齐次解[1]。因此，需要使用阶跃函数作为初始条件，以扰动稳态和促进方程组演化。

$$u(x, 0) = \begin{bmatrix} n_0 \\ c_0 \end{bmatrix},$$

$$u(x, 0) = \begin{cases} 1.05u_1 & 0.3 \leq x \leq 0.6 \\ 1.0005u_2 & 0.3 \leq x \leq 0.6 \end{cases}$$

对应的函数是

```
function u0 = angioic(x)
u0 = [1; 0.5];
if x >= 0.3 && x <= 0.6
    u0(1) = 1.05 * u0(1);
    u0(2) = 1.0005 * u0(2);
end
end
```

编写边界条件代码

现在，编写计算以下边界条件的函数

$$\frac{\partial}{\partial x} n(0, t) = \frac{\partial}{\partial x} n(1, t) = 0,$$

$$\frac{\partial}{\partial x} c(0, t) = \frac{\partial}{\partial x} c(1, t) = 0.$$

对于在区间 $a \leq x \leq b$ 上提出的问题，边界条件应用于所有 t 以及 $x = a$ 或 $x = b$ 。求解器所需的标准形式是

$$p(x, t, u) + q(x, t) f \left(x, t, u, \frac{\partial u}{\partial x} \right) = 0.$$

对于 $x = 0$ ，边界条件方程为

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} p(x, t, u) + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} d \frac{\partial n}{\partial x} - a n \frac{\partial c}{\partial x} \\ \frac{\partial c}{\partial x} \end{bmatrix} = 0.$$

因此系数为：

- $p_L(x, t, u) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$,
- $q_L(x, t) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

对于 $x = 1$ ，边界条件是相同的，因此 $p_R(x, t, u) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ 且 $q_R(x, t) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ 。

边界函数应使用函数签名 `[pl,ql,pr,qr] = angiobc(xl,ul,xr,ur,t)`，其中：

- 对于左边界，输入 `xl` 和 `ul` 对应于 u 和 x 。
- 对于右边界，输入 `xr` 和 `ur` 对应于 u 和 x 。
- t 是独立的时间变量。

- 对于左边界，输出 **pl** 和 **ql** 对应于 $p_L(x, t, u)$ 和 $q_L(x, t)$ (对于此问题， $x = 0$)。
- 对于右边界，输出 **pr** 和 **qr** 对应于 $p_R(x, t, u)$ 和 $q_R(x, t)$ (对于此问题， $x = 1$)。

此示例中的边界条件由以下函数表示：

```
function [pl,ql,pr,qr] = angiobc(xl,ul,xr,ur,t)
pl = [0; 0];
ql = [1; 1];
pr = pl;
qr = ql;
end
```

选择解网格

要了解方程的限制行为，需要很长的时间区间，因此使用 10 个位于区间 $0 \leq t \leq 200$ 中的点。此外，在 $0 \leq x \leq 1$ 区间内， $c(x, t)$ 的限值分布仅变化约 0.1%，因此具有 50 个点的相对精细的空间网格是合适的。

```
x = linspace(0,1,50);
t = linspace(0,200,10);
```

求解方程

最后，使用对称性值 m 、PDE 方程、初始条件、边界条件以及 x 和 t 的网格来求解方程。

```
m = 0;
sol = pdepe(m,@angiopde,@angioic,@angiobc,x,t);
```

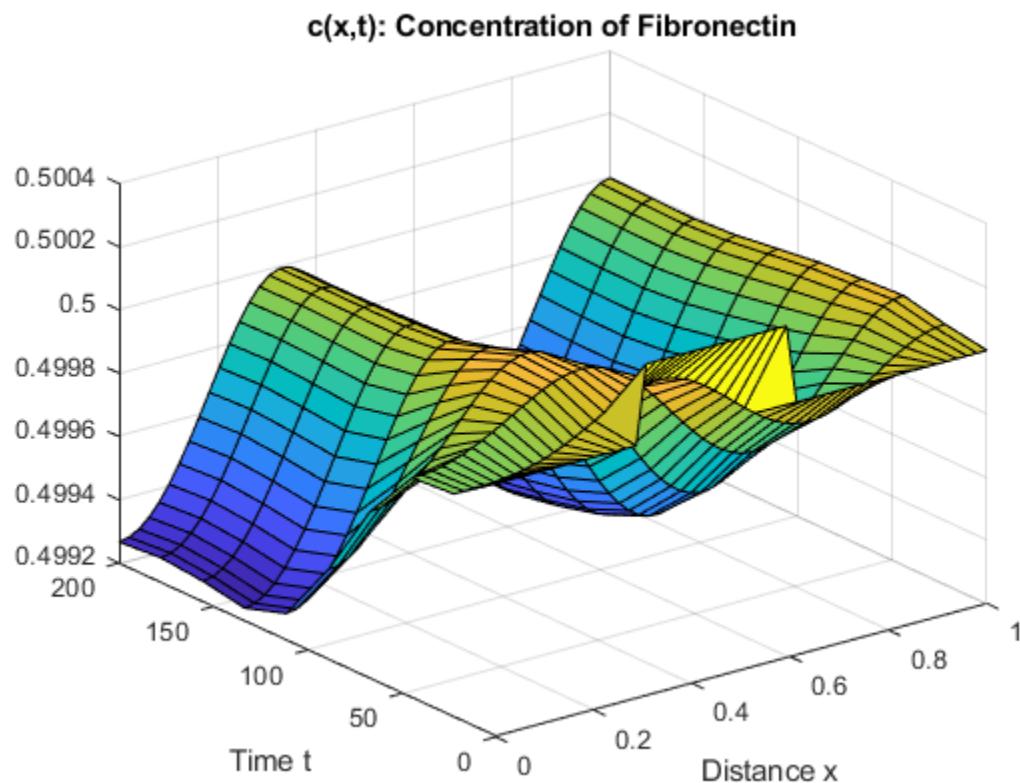
pdepe 以三维数组 **sol** 形式返回解，其中 **sol(i,j,k)** 是在 $t(i)$ 和 $x(j)$ 处计算的解 u_k 的第 k 个分量的逼近值。将解分量提取到单独的变量中。

```
n = sol(:,:,1);
c = sol(:,:,2);
```

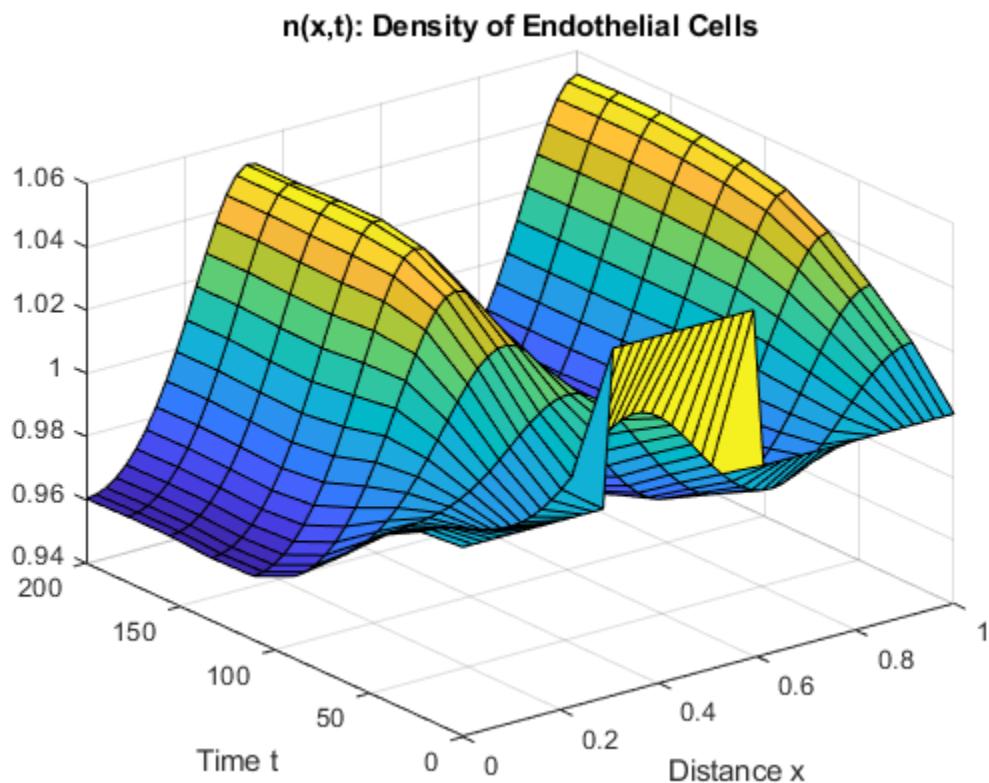
对解进行绘图

创建基于所选的 x 和 t 网格点绘制的解分量 n 和 c 的曲面图。

```
surf(x,t,c)
title('c(x,t): Concentration of Fibronectin')
xlabel('Distance x')
ylabel('Time t')
```

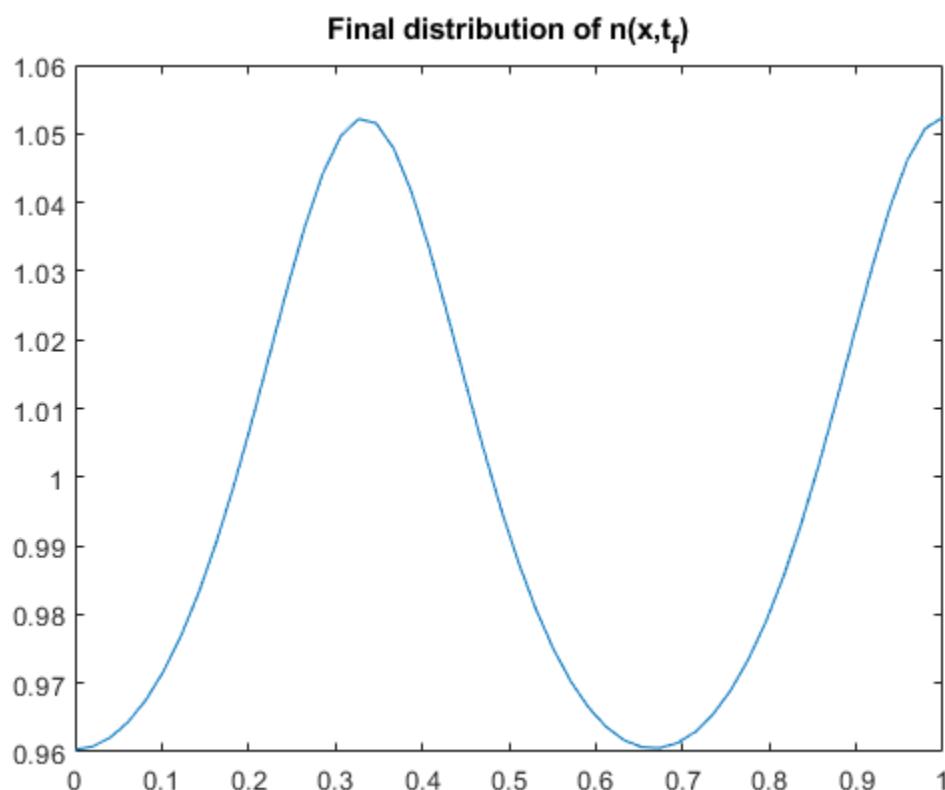


```
surf(x,t,n)
title('n(x,t): Density of Endothelial Cells')
xlabel('Distance x')
ylabel('Time t')
```

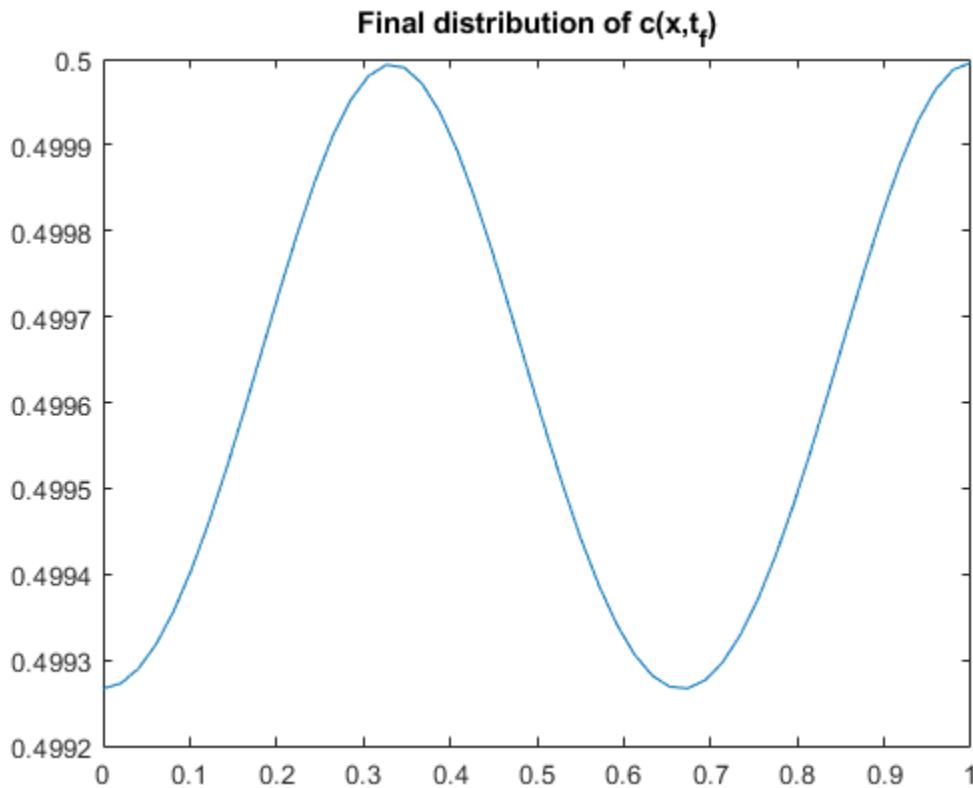


现在，仅绘制在 $t_f = 200$ 处的解的最终分布。这些图对应于 [1] 中的图 3 和 4。

```
plot(x,n(end,:))
title('Final distribution of n(x,t_f)')
```



```
plot(x,c(end,:))  
title('Final distribution of c(x,t_f)')
```



参考

[1] Humphreys, M.E. and M.A.J. Chaplain."A mathematical model of the first steps of tumour-related angiogenesis: Capillary sprout formation and secondary branching." IMA Journal of Mathematics Applied in Medicine & Biology, 13 (1996), pp. 73-98.

局部函数

此处列出 PDE 求解器 pdepe 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```

function [c,f,s] = angiopde(x,t,u,dudx) % Equation to solve
d = 1e-3;
a = 3.8;
S = 3;
r = 0.88;
N = 1;

c = [1; 1];
f = [d*dudx(1) - a*u(1)*dudx(2)
      dudx(2)];
s = [S*r*u(1)*(N - u(1));
      S*(u(1)/(u(1) + 1) - u(2))];
end
%
function u0 = angioic(x) % Initial Conditions
u0 = [1; 0.5];

```

```
if x >= 0.3 && x <= 0.6
    u0(1) = 1.05 * u0(1);
    u0(2) = 1.0005 * u0(2);
end
end
%
function [pl,ql,pr,qr] = angiobc(xl,ul,xr,ur,t) % Boundary Conditions
pl = [0; 0];
ql = [1; 1];
pr = pl;
qr = ql;
end
%
```

另请参阅

[pdepe](#)

详细信息

- “求解偏微分方程” (第 13-2 页)
- “求解单个 PDE” (第 13-9 页)
- “求解 PDE 方程组” (第 13-30 页)

时滞微分方程 (DDE)

- “解算时滞微分方程” (第 14-2 页)
- “具有常时滞的 DDE” (第 14-5 页)
- “具有状态相关时滞的 DDE” (第 14-8 页)
- “具有不连续性的心血管模型 DDE” (第 14-12 页)
- “中立型 DDE” (第 14-17 页)
- “中立型的初始值 DDE” (第 14-21 页)

解算时滞微分方程

时滞微分方程 (DDE) 是当前时间的解与过去时间的解相关的常微分方程。该时滞可以固定不变、与时间相关、与状态相关或与导数相关。要开始积分，通常必须提供历史解，以便求解器可以获取初始积分点之前的时间的解。

常时滞 DDE

具有常时滞的微分方程组的形式如下：

$$y'(t) = f(t, y(t), y(t - \tau_1), \dots, y(t - \tau_k)).$$

此处， t 为自变量， y 为因变量的列向量，而 y' 表示 y 关于 t 的一阶导数。时滞 τ_1, \dots, τ_k 是正常量。

dde23 函数用于求解具有历史解 $y(t) = S(t)$ (其中 $t < t_0$) 的常时滞 DDE。

DDE 的解通常是连续的，但其导数不连续。**dde23** 函数跟踪低阶导数的不连续性，并使用 **ode23** 使用的同一显式 Runge-Kutta (2,3) 对和插值求微分方程的积分。对于大于时滞的步长而言，Runge-Kutta 公式是隐式的。当 $y(t)$ 足够平滑以证明此大小的步长时，使用预测-校正迭代法计算隐式公式。

时间相关和状态相关的 DDE

常时滞 DDE 是一种特殊情况，更为一般的 DDE 形式为：

$$y'(t) = f(t, y(t), y(dy_1), \dots, y(dy_p)).$$

时间相关和状态相关的 DDE 涉及可能依赖于时间 t 和状态 y 的时滞 dy_1, \dots, dy_k 。时滞 $dy_j(t, y)$ 必须满足 $dy_j(t, y) \leq t$ (在区间 $[t_0, t_f]$ 上，其中 $t_0 < t_f$)。

ddesd 函数用于求具有历史解 $y(t) = S(t)$ (其中 $t < t_0$) 的时间相关和状态相关 DDE 的解 $y(t)$ 。**ddesd** 函数使用标准的四级、四阶显式 Runge-Kutta 法来求积分，并它控制自然插值的余值大小。它使用迭代来采用超过时滞的步长。

中立型 DDE

中立型的时滞微分方程涉及在 y' 以及 y 中的时滞：

$$y'(t) = f(t, y(t), y(dy_1), \dots, y(dy_p), y(dyp_1), \dots, y(dyp_q)).$$

解中的时滞必须满足 $dy_i(t, y) \leq t$ 。一阶导数的时滞必须满足 $dyp_j(t, y) < t$ ，以便 y' 不显示在方程两端。

ddensd 函数使用时间相关和状态相关 DDE 来逼近中立型 DDE，从而对其求解：

$$y'(t) = f(t, y(t), y(dy_1), \dots, y(dy_p)).$$

有关详细信息，请参阅 Shampine[1]。

计算特定点的解

使用 **deval** 函数和任何 DDE 求解器的输出来计算积分区间中的特定点处的解。例如， $y = \text{deval}(\text{sol}, 0.5 * (\text{sol.x}(1) + \text{sol.x}(\text{end})))$ 计算积分区间中点处的解。

历史解和初始值

对 DDE 求解时，将在区间 $[t_0, t_f]$ (其中 $t_0 < t_f$) 上来逼近解。DDE 表明 $y(t)$ 如何依赖于 t 之前的时间的解 (及其可能的导数) 的值。例如，具有常时滞时， $y'(t_0)$ 依赖于 $y(t_0 - \tau_1), \dots, y(t_0 - \tau_k)$ ，其中 τ_j 为正常量。因此， $[t_0, t_k]$ 上的解依赖于其在 $t \leq t_0$ 处具有的值。必须使用历史解函数 $y(t) = S(t)$ (其中 $t < t_0$) 定义这些值。

DDE 中的不连续性

如果问题具有不连续性，最好使用 options 结构体将其传递给求解器。为此，请使用 `ddeset` 创建一个 options 结构体以包含问题中的不连续性。

`options` 结构体中有三个属性可用于指定不连续性：`InitialY`、`Jumps` 和 `Events`。选择的属性取决于不连续性的位置和特性。

不连续性的特性	属性	注释
在初始值 $t = t_0$ 处	<code>InitialY</code>	初始值 $y(t_0)$ 通常是历史解函数返回的值 $S(t_0)$ ，也就是说，解在初始点连续。如果不属于此种情况，请使用 <code>InitialY</code> 属性提供一个不同的初始值。
在历史解 (即 $t < t_0$ 的解) 或在 $t > t_0$ 的方程系数中	<code>Jumps</code>	在向量中提供不连续处的已知位置 t ，以作为 <code>Jumps</code> 属性的值。仅适用于 <code>dde23</code> 。
与状态相关	<code>Events</code>	<code>dde23</code> 、 <code>ddesd</code> 和 <code>ddensd</code> 使用您提供的事件函数来查找这些不连续的位置。当求解器查找这种不连续的位置时，请重新启动积分以便继续。将当前积分的解结构体指定为新积分的历史解。求解器在每次重新启动之后扩展解结构体的每个元素，以使最终结构体为整个积分区间提供解。如果新问题与解变化相关，请使用 <code>InitialY</code> 属性指定新集成的初始值。

不连续性传播

通常，解的一阶导数在初始点处具有跳跃性。这是因为历史解函数 $S(t)$ 的一阶导数通常在此点处不满足 DDE。当时滞为常量时，则 $y(t)$ 的任何导数中的以间距 τ_1, \dots, τ_k 传播不连续点。如果时滞不为常量，则不连续性的传播更复杂。对于“常时滞 DDE”(第 14-2 页)或“时间相关和状态相关的 DDE”(第 14-2 页)形式的中立型 DDE，每次传播时，不连续性会出现在高一阶的导数中。在这种意义上，解会随着积分的进行而变得更圆滑。“中立型 DDE”(第 14-2 页)中指定的形式的中立型 DDE 的解在性质上不同。解的不连续性不会传播到高阶的导数。特别是， $y'(t)$ 中 t_0 处的典型跳跃作为 $y(t)$ 中的跳跃传播到整个 $[t_0, t_f]$ 。

DDE 示例和文件

可使用几个示例文件作为求解最常见 DDE 问题的绝佳起点。要方便地查看和运行示例，可以使用 **Differential Equations Examples App**。要运行此 App，请键入

`odeexamples`

要打开单独的示例文件进行编辑，请键入

`edit exampleFileName.m`

要运行示例，请键入

`exampleFileName`

下表包含可用的 DDE 示例文件及其使用的求解器和选项的列表。

示例文件	使用的求解器	指定的选项	说明	示例链接
<code>ddex1</code>	<code>dde23</code>	—	具有常历史解的 DDE	“具有常时滞的 DDE” (第 14-5 页)
<code>ddex2</code>	<code>dde23</code>	• 'Jumps'	具有不连续性的 DDE	“具有不连续性的 心血管模型 DDE” (第 14-12 页)
<code>ddex3</code>	<code>ddesd</code>	—	具有状态相关时滞的 DDE	“具有状态相关时 滞的 DDE” (第 14-8 页)
<code>ddex4</code>	<code>ddensd</code>	—	带有两个时滞的中立型 DDE	“中立型 DDE” (第 14-17 页)
<code>ddex5</code>	<code>ddensd</code>	—	具有初始值的中立型 DDE	“中立型的初始值 DDE” (第 14-21 页)

参考

- [1] Shampine, L.F. "Dissipative Approximations to Neutral DDEs." *Applied Mathematics & Computation*, Vol. 203, 2008, pp. 641–648.

另请参阅

`dde23` | `ddensd` | `ddesd` | `ddeset`

详细信息

- “具有常时滞的 DDE” (第 14-5 页)
- “具有状态相关时滞的 DDE” (第 14-8 页)

具有常时滞的 DDE

以下示例说明如何使用 **dde23** 对具有常时滞的 DDE (时滞微分方程) 方程组求解。

方程组为：

$$\begin{aligned}y_1'(t) &= y_1(t - 1) \\y_2'(t) &= y_1(t - 1) + y_2(t - 0.2) \\y_3'(t) &= y_2(t).\end{aligned}$$

$t \leq 0$ 的历史解函数是常量 $y_1(t) = y_2(t) = y_3(t) = 1$ 。

方程中的时滞仅存在于 y 项中，并且时滞本身是常量，因此各方程构成常时滞方程组。

要在 MATLAB 中求解此方程组，您需要先编写方程组、时滞和历史解的代码，然后再调用时滞微分方程求解器 **dde23**，该求解器适用于具有常时滞的方程组。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写时滞代码

首先，创建一个向量来定义方程组中的时滞。此方程组有两种不同时滞：

- 在第一个分量 $y_1(t - 1)$ 中时滞为 1。
- 在第二个分量 $y_2(t - 0.2)$ 中时滞为 0.2。

dde23 接受时滞的向量参数，其中每个元素是一个分量的常时滞。

```
lags = [1 0.2];
```

编写方程代码

现在，创建一个函数来编写方程的代码。此函数应具有签名 **dydt = ddefun(t,y,Z)**，其中：

- **t** 是时间（自变量）。
- **y** 是解（因变量）。
- **Z(:,j)** 用于逼近时滞 $y(t - \tau_j)$ ，其中常时滞 τ_j 由 **lags(j)** 给定。

求解器会自动将这些输入传递给该函数，但是变量名称决定如何编写方程代码。在这种情况下：

- **Z(:,1)** $\rightarrow y_1(t - 1)$
- **Z(:,2)** $\rightarrow y_2(t - 0.2)$

```
function dydt = ddex1de(t,y,Z)
    ylag1 = Z(:,1);
    ylag2 = Z(:,2);

    dydt = [ylag1(1);
            ylag1(1)+ylag2(2);
            y(2)];
end
```

注意：所有函数都作为局部函数包含在示例的末尾。

编写历史解代码

接下来，创建一个函数来定义历史解。历史解是时间 $t \leq t_0$ 的解。

```
function s = history(t)
    s = ones(3,1);
end
```

求解方程

最后，定义积分区间 $[t_0 t_f]$ 并使用 `dde23` 求解器对 DDE 求解。

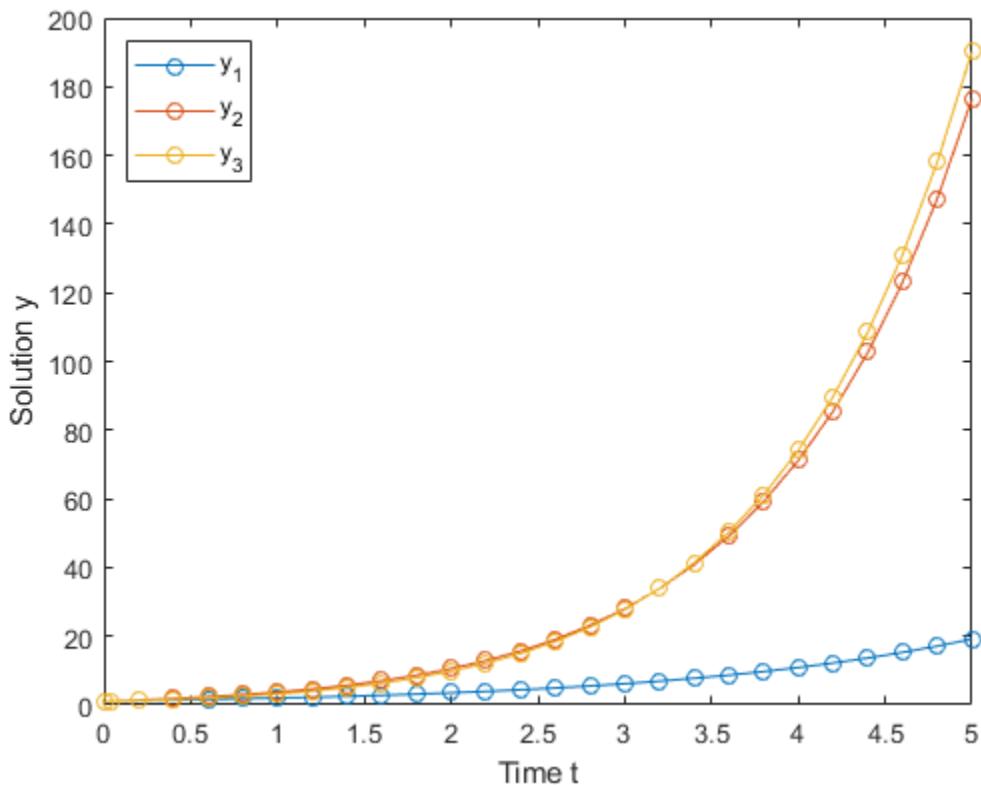
```
tspan = [0 5];
sol = dde23(@ddefun, lags, @history, tspan);
```

对解进行绘图

解结构体 `sol` 具有字段 `sol.x` 和 `sol.y`，这两个字段包含求解器在这些时间点所用的内部时间步和对应的解。（如果您需要在特定点的解，可以使用 `deval` 来计算在特定点的解。）

绘制三个解分量对时间的图。

```
plot(sol.x,sol.y,'-o')
xlabel('Time t');
ylabel('Solution y');
legend('y_1','y_2','y_3','Location','NorthWest');
```



局部函数

此处列出了 DDE 求解器 `dde23` 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```
function dydt = ddefun(t,y,Z) % equation being solved
    ylag1 = Z(:,1);
    ylag2 = Z(:,2);

    dydt = [ylag1(1);
             ylag1(1)+ylag2(2);
             y(2)];
end
%-----
function s = history(t) % history function for t <= 0
    s = ones(3,1);
end
%
```

另请参阅

[dde23](#) | [ddensd](#) | [ddesd](#) | [deval](#)

详细信息

- “解算时滞微分方程” (第 14-2 页)
- “具有状态相关时滞的 DDE” (第 14-8 页)

具有状态相关时滞的 DDE

以下示例说明如何使用 `ddesd` 对具有状态相关时滞的 DDE (时滞微分方程) 方程组求解。Enright 和 Hayashi [1] 将此 DDE 方程组用作测试问题。

方程组为：

$$y_1'(t) = y_2(t),$$

$$y_2'(t) = -y_2\left(e^1 - y_2(t)\right) \cdot y_2(t)^2 \cdot e^{1-y_2(t)}.$$

$t \leq 0.1$ 的历史解函数是解析解

$$y_1(t) = \log(t),$$

$$y_2(t) = \frac{1}{t}.$$

方程中的时滞仅出现在 y 项中。时滞仅取决于第二个分量 $y_2(t)$ 的状态，因此这些方程构成状态相关时滞方程组。

要在 MATLAB 中求解此方程组，您需要先编写方程组、时滞和历史解的代码，然后再调用时滞微分方程求解器 `ddesd`，该求解器适用于具有状态相关时滞的方程组。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

编写时滞代码

首先，编写一个函数来定义方程组中的时滞。此方程组中唯一存在的时滞位于项 $-y_2\left(e^1 - y_2(t)\right)$ 中。

```
function d = dely(t,y)
d = exp(1 - y(2));
end
```

注意：所有函数都作为局部函数包含在示例的末尾。

编写方程代码

现在，创建一个函数来编写方程的代码。此函数应具有签名 `dydt = ddefun(t,y,Z)`，其中：

- t 是时间（自变量）。
- y 是解（因变量）。
- $Z(n,j)$ 对时滞 $y_n(d(j))$ 求近似值，其中时滞 $d(j)$ 由 `dely(t,y)` 的分量 j 给出。

求解器会自动将这些输入传递给该函数，但是变量名称决定如何编写方程代码。在这种情况下：

$$\bullet \quad Z(2,1) \rightarrow y_2\left(e^1 - y_2(t)\right)$$

```
function dydt = ddefun(t,y,Z)
dydt = [y(2);
        -Z(2,1)*y(2)^2*exp(1 - y(2))];
end
```

编写历史解代码

接下来，创建一个函数来定义历史解。历史解是时间 $t \leq t_0$ 的解。

```
function v = history(t) % history function for t < t0
v = [log(t);
      1./t];
end
```

求解方程

最后，定义积分区间 $[t_0 \ t_f]$ 并使用 **ddesd** 求解器对 DDE 求解。

```
tspan = [0.1 5];
sol = ddesd(@ddefun, @dely, @history, tspan);
```

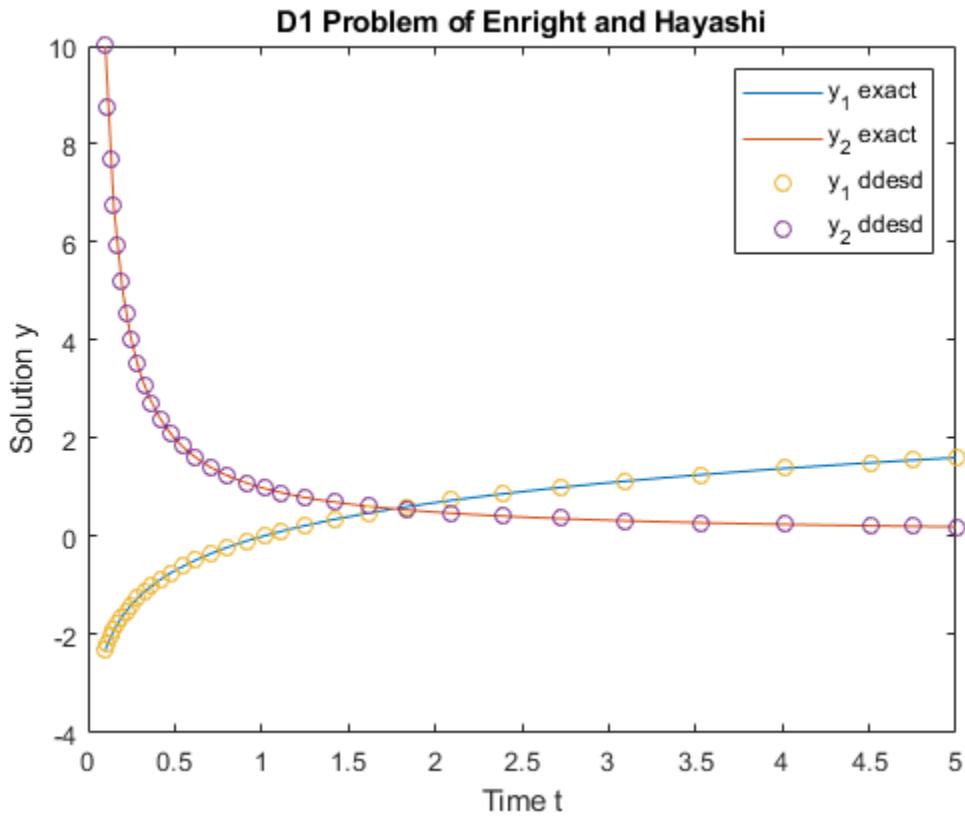
对解进行绘图

解结构体 **sol** 具有字段 **sol.x** 和 **sol.y**，这两个字段包含求解器在这些时间点所用的内部时间步和对应的解。（如果您需要在特定点的解，可以使用 **deval** 来计算在特定点的解。）

使用历史解函数绘制两个解分量对时间的图，以计算积分区间内的解析解来进行比较。

```
ta = linspace(0.1,5);
ya = history(ta);

plot(ta,ya,sol.x,sol.y,'o')
legend('y_1 exact','y_2 exact','y_1 ddesd','y_2 ddesd')
xlabel('Time t')
ylabel('Solution y')
title('D1 Problem of Enright and Hayashi')
```



局部函数

此处列出了 DDE 求解器 `ddesd` 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```

function dydt = ddefun(t,y,Z) % equation being solved
dydt = [y(2);
        -Z(2,1).*y(2)^2.*exp(1 - y(2))];
end
%
function d = delay(t,y) % delay for y
d = exp(1 - y(2));
end
%
function v = history(t) % history function for t < t0
v = [log(t);
      1./t];
end
%
```

参考

- [1] Enright, W.H. and H. Hayashi. "The Evaluation of Numerical Software for Delay Differential Equations." In Proceedings of the IFIP TC2/WG2.5 working conference on Quality of numerical

software: assessment and enhancement.(R.F. Boisvert, ed.).London, UK:Chapman & Hall, Ltd., pp. 179-193.

另请参阅

[dde23](#) | [ddensd](#) | [ddesd](#) | [deval](#)

详细信息

- “解算时滞微分方程” (第 14-2 页)
- “具有不连续性的心血管模型 DDE” (第 14-12 页)

具有不连续性的心血管模型 DDE

此示例说明如何使用 **dde23** 对具有不连续导数的心血管模型求解。此示例最初由 Ottesen [1] 提出。

方程组为：

$$\dot{P}_a(t) = -\frac{1}{c_a R} P_a(t) + \frac{1}{c_a R} P_v(t) + \frac{1}{c_a} V_{\text{str}}(P_a^\tau(t)) H(t)$$

$$\dot{P}_v(t) = \frac{1}{c_v R} P_a(t) - \left(\frac{1}{c_v R} + \frac{1}{c_v r} \right) P_v(t)$$

$$\dot{H}(t) = \frac{\alpha_H T_s}{1 + \gamma_H T_p} - \beta_H T_p.$$

T_s 和 T_p 的项分别是同一方程在有时滞和没有时滞状态下的变体。 P_a^τ 和 P_a 分别代表在有时滞和没有时滞状态下的平均动脉压。

$$T_s = \frac{1}{1 + \left(\frac{P_a^\tau}{\alpha_s} \right)^{\beta_s}}$$

$$T_p = \frac{1}{1 + \left(\frac{P_a}{\alpha_p} \right)^{-\beta_p}}.$$

此问题有许多物理参数：

- 动脉顺应性 $c_a = 1.55 \text{ ml/mmHg}$
- 静脉顺应性 $c_v = 519 \text{ ml/mmHg}$
- 外周阻力 $R = 1.05(0.84) \text{ mmHg s/ml}$
- 静脉流出阻力 $r = 0.068 \text{ mmHg s/ml}$
- 心博量 $V_{\text{str}} = 67.9(77.9) \text{ ml}$
- 典型平均动脉压 $P_0 = 93 \text{ mmHg}$
- $\alpha_0 = \alpha_s = \alpha_p = 93(121) \text{ mmHg}$
- $\alpha_H = 0.84 \text{ sec}^{-2}$
- $\beta_0 = \beta_s = \beta_p = 7$
- $\beta_H = 1.17$
- $\gamma_H = 0$

该方程组受外周压的巨大影响，外周压会从 $R = 1.05$ 急剧减少到 $R = 0.84$ ，从 $t = 600$ 处开始。因此，该方程组在 $t = 600$ 处的低阶导数具有不连续性。

常历史解由以下物理参数定义

$$P_a = P_0, \quad P_v(t) = \frac{1}{1 + \frac{R}{r}} P_0, \quad H(t) = \frac{1}{RV_{\text{str}}} \frac{1}{1 + \frac{r}{R}} P_0.$$

要在 MATLAB 中求解此方程组，您需要先编写方程组、参数、时滞和历史解的代码，然后再调用时滞微分方程求解器 **dde23**，该求解器适用于具有常时滞的方程组。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的命名文件保存在 MATLAB 路径上的目录中。

定义物理参数

首先，将问题的物理参数定义为结构体中的字段。

```
p.ca = 1.55;
p.cv = 519;
p.R = 1.05;
p.r = 0.068;
p.Vstr = 67.9;
p.alpha0 = 93;
p.alphas = 93;
p.alphap = 93;
p.alphaH = 0.84;
p.beta0 = 7;
p.betas = 7;
p.betap = 7;
p.betaH = 1.17;
p.gammaH = 0;
```

编写时滞代码

接下来，创建变量 **tau** 来表示项 $P_a^\tau(t) = P_a(t - \tau)$ 的方程中的常时滞 τ 。

```
tau = 4;
```

编写方程代码

现在，创建一个函数来编写方程的代码。此函数应具有签名 **dydt = ddefun(t,y,Z,p)**，其中：

- **t** 是时间（自变量）。
- **y** 是解（因变量）。
- **Z(n,j)** 对时滞 $y_n(d(j))$ 求近似值，其中时滞 $d(j)$ 由 **dely(t,y)** 的分量 **j** 给出。
- **p** 是可选的第四个输入，用于传入参数值。

求解器自动将前三个输入传递给函数，变量名称决定如何编写方程代码。调用求解器时，参数结构体 **p** 将传递给函数。在本例中，时滞表示为：

- $Z(:,1) \rightarrow P_a(t - \tau)$

```
function dydt = ddefun(t,y,Z,p)
if t <= 600
    p.R = 1.05;
else
    p.R = 0.21 * exp(600-t) + 0.84;
end
ylag = Z(:,1);
Patau = ylag(1);
Paoft = y(1);
Pvoft = y(2);
Hoft = y(3);

dPadt = - (1 / (p.ca * p.R)) * Paoft ...
```

```

+ (1/(p.ca * p.R)) * Pvoft ...
+ (1/p.ca) * p.Vstr * Hoft;

dPvdt = (1 / (p.cv * p.R)) * Paoft...
- ( 1 / (p.cv * p.R)...
+ 1 / (p.cv * p.r) ) * Pvoft;

Ts = 1 / ( 1 + (Patau / p.alphas)^p.betas );
Tp = 1 / ( 1 + (p.alphap / Paoft)^p.betap );

dHdt = (p.alphaH * Ts) / (1 + p.gammaH * Tp) ...
- p.betaH * Tp;

dydt = [dPadt; dPvdt; dHdt];
end

```

注意：所有函数都作为局部函数包含在示例的末尾。

编写历史解代码

接下来，创建一个向量来定义三个分量 P_a 、 P_v 和 H 的常历史解。历史解是时间 $t \leq t_0$ 的解。

```

P0 = 93;
Paval = P0;
Pvval = (1 / (1 + p.R/p.r)) * P0;
Hval = (1 / (p.R * p.Vstr)) * (1 / (1 + p.r/p.R)) * P0;
history = [Paval; Pvval; Hval];

```

求解方程

使用 `ddeset` 来指定在 $t = 600$ 处存在不连续性。最后，定义积分区间 $[t_0 \ t_f]$ 并使用 `dde23` 求解器对 DDE 求解。使用匿名函数指定 `ddefun` 以传入参数结构体 `p`。

```

options = ddeset('Jumps',600);
tspan = [0 1000];
sol = dde23(@(t,y,Z) ddefun(t,y,Z,p), tau, history, tspan, options);

```

对解进行绘图

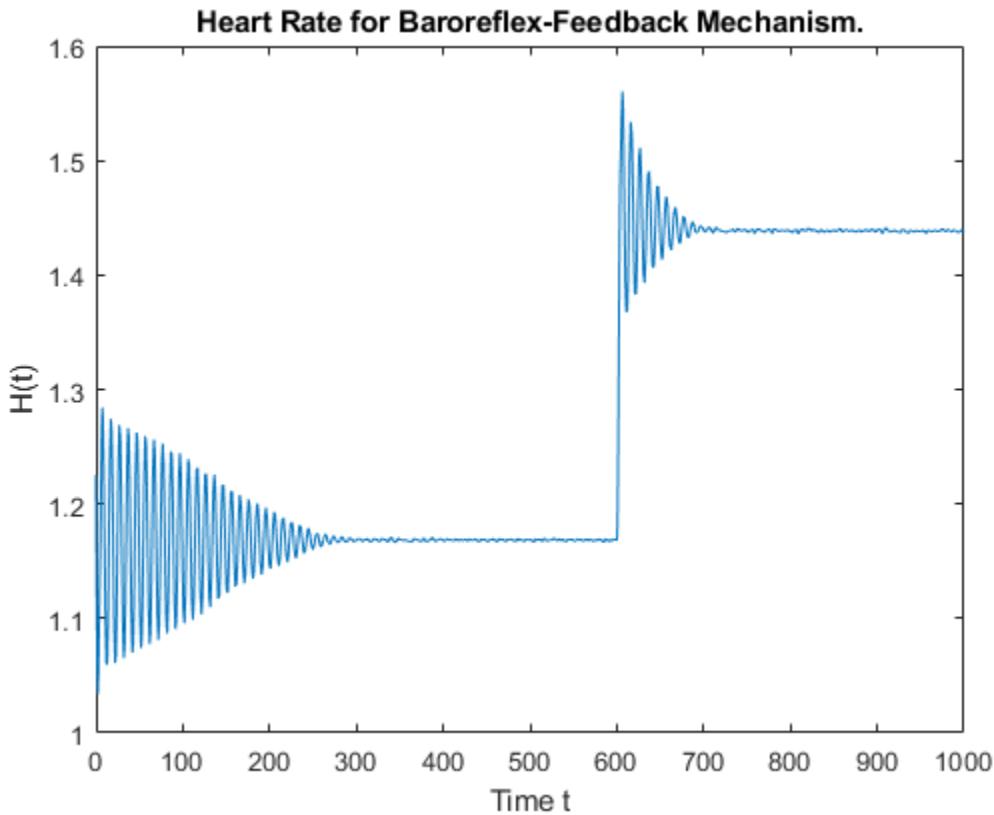
解结构体 `sol` 具有字段 `sol.x` 和 `sol.y`，这两个字段包含求解器在这些时间点所用的内部时间步和对应的解。（如果您需要在特定点的解，可以使用 `deval` 来计算在特定点的解。）

绘制第三个解分量（心率）对时间的图。

```

plot(sol.x,sol.y(3,:))
title('Heart Rate for Baroreflex-Feedback Mechanism.')
xlabel('Time t')
ylabel('H(t)')

```



局部函数

此处列出了 DDE 求解器 `dde23` 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```
function dydt = ddefun(t,y,Z,p) % equation being solved
    if t <= 600
        p.R = 1.05;
    else
        p.R = 0.21 * exp(600-t) + 0.84;
    end
    ylag = Z(:,1);
    Patau = ylag(1);
    Paoft = y(1);
    Pvof = y(2);
    Hoft = y(3);

    dPadt = - (1 / (p.ca * p.R)) * Paoft ...
        + (1/(p.ca * p.R)) * Pvof ...
        + (1/p.ca) * p.Vstr * Hoft;

    dPvdt = (1 / (p.cv * p.R)) * Paoft...
        - ( 1 / (p.cv * p.R)...
        + 1 / (p.cv * p.r) ) * Pvof;

    Ts = 1 / ( 1 + (Patau / p.alphas)^p.betas );
    Tp = 1 / ( 1 + (p.alphap / Paoft)^p.betap );
```

```
dHdt = (p.alphaH * Ts) / (1 + p.gammaH * Tp) ...
    - p.betaH * Tp;
dydt = [dPadt; dPvdt; dHdt];
end
```

参考

[1] Ottesen, J. T. "Modelling of the Baroreflex-Feedback Mechanism with Time-Delay." *J. Math.Biol.* Vol. 36, Number 1, 1997, pp. 41–63.

另请参阅

[dde23](#) | [ddensd](#) | [ddesd](#) | [deval](#)

详细信息

- “解算时滞微分方程” (第 14-2 页)
- “中立型 DDE” (第 14-17 页)

中立型 DDE

以下示例说明如何使用 `ddensd` 求解中立型 DDE（时滞微分方程），其中时滞出现在导数项中。此问题最初由 Paul [1] 提出。

方程是：

$$y'(t) = 1 + y(t) - 2y\left(\frac{t}{2}\right)^2 - y'(t - \pi).$$

在 $t \leq 0$ 时，历史解函数是 $y(t) = \cos(t)$ 。

由于该方程在 y' 项中存在时滞，因此该方程称为中立型 DDE。如果时滞仅出现在 y 项中，则根据时滞的形式，方程将是常时滞或状态相关 DDE。

要在 MATLAB 中求解此方程，您需要先编写方程、时滞和历史解的代码，然后再调用时滞微分方程求解器 `ddensd`。您可以将这些作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的文件保存在 MATLAB 路径上的目录中。

编写时滞代码

首先，编写函数来定义方程中的时滞。方程中具有时滞的首项是 $y\left(\frac{t}{2}\right)$ 。

```
function dy = dely(t,y)
    dy = t/2;
end
```

方程中具有时滞的另一个项是 $y'(t - \pi)$ 。

```
function dyp = delyp(t,y)
    dyp = t-pi;
end
```

在此示例中， y 和 y' 分别仅有一个时滞。如果有更多时滞，则您可以将它们添加到这些相同的函数文件中，这样函数将返回向量而不是标量。

注意：所有函数都作为局部函数包含在示例的末尾。

编写方程代码

现在，创建一个函数来编写方程代码。此函数应具有签名 `yp = ddefun(t,y,ydel,ypdel)`，其中：

- t 是时间（自变量）。
- y 是解（因变量）。
- $ydel$ 包含 y 的时滞。
- $ypdel$ 包含 $y' = \frac{dy}{dt}$ 的时滞。

求解器会自动将这些输入传递给该函数，但是变量名称决定如何编写方程代码。在这种情况下：

- $ydel \rightarrow y\left(\frac{t}{2}\right)$
- $ypdel \rightarrow y'(t - \pi)$

```
function yp = ddefun(t,y,ydel,ypdel)
    yp = 1 + y - 2*ydel^2 - ypdel;
end
```

编写历史解代码

接下来，创建一个函数来定义历史解。历史解是时间 $t \leq t_0$ 的解。

```
function y = history(t)
    y = cos(t);
end
```

求解方程

最后，定义积分区间 $[t_0 \ t_f]$ 并使用 **ddensd** 求解器对 DDE 求解。

```
tspan = [0 pi];
sol = ddensd(@ddefun, @dely, @delyp, @history, [0,pi]);
```

对解进行绘图

解结构体 **sol** 具有字段 **sol.x** 和 **sol.y**，这两个字段包含求解器在这些时间点所用的内部时间步和对应的解。但是，您可以使用 **deval** 计算在特定点的解。

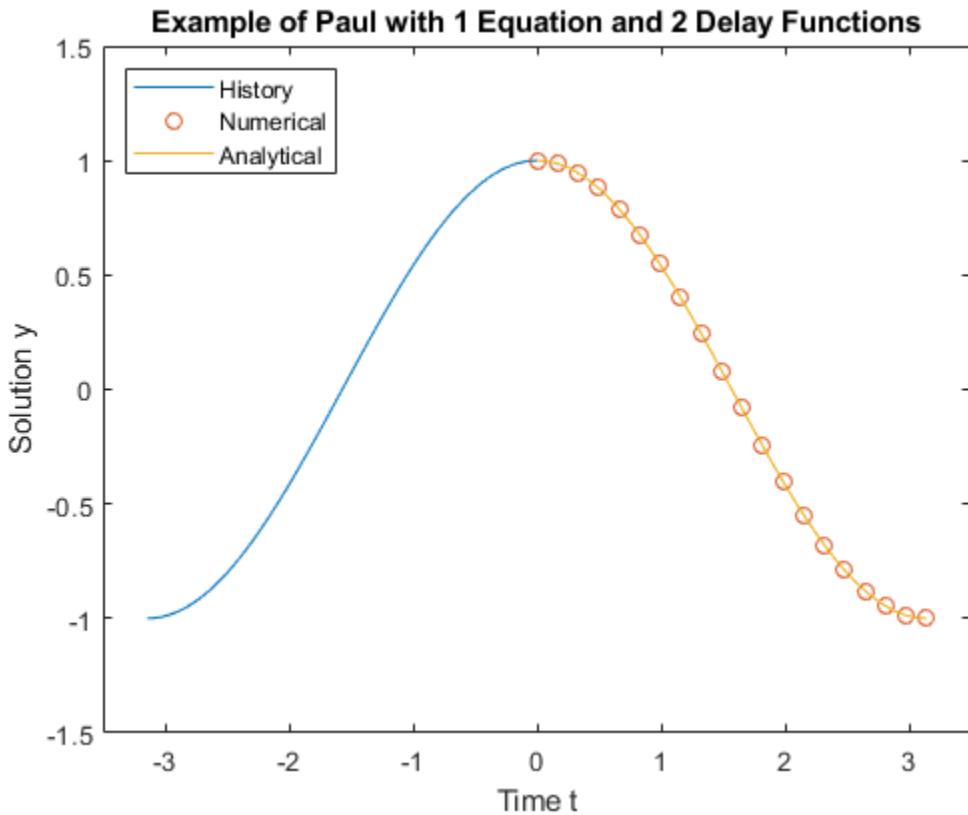
在 0 和 **pi** 之间以 20 个等距点计算解。

```
tn = linspace(0,pi,20);
yn = deval(sol,tn);
```

绘制计算解和历史解对解析解的图。

```
th = linspace(-pi,0);
yh = history(th);
ta = linspace(0,pi);
ya = cos(ta);

plot(th,yh,tn,yn,'o',ta,ya)
legend('History','Numerical','Analytical','Location','NorthWest')
xlabel('Time t')
ylabel('Solution y')
title('Example of Paul with 1 Equation and 2 Delay Functions')
axis([-3.5 3.5 -1.5 1.5])
```



局部函数

此处列出了 DDE 求解器 `ddensd` 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```

function yp = ddefun(t,y,ydel,ypdel) % equation being solved
    yp = 1 + y - 2*ydel^2 - ypdel;
end
%
function dy = delay(t,y) % delay for y
    dy = t/2;
end
%
function dyp = delyp(t,y) % delay for y'
    dyp = t-pi;
end
%
function y = history(t) % history function for t < 0
    y = cos(t);
end
%
```

参考

[1] Paul, C.A.H. "A Test Set of Functional Differential Equations." Numerical Analysis Reports.No. 243. Manchester, UK: Math Department, University of Manchester, 1994.

另请参阅

[dde23](#) | [ddensd](#) | [ddesd](#) | [deval](#)

详细信息

- “解算时滞微分方程” (第 14-2 页)
- “中立型的初始值 DDE” (第 14-21 页)

中立型的初始值 DDE

以下示例说明如何使用 **ddensd** 求解具有时间相关时滞的初始值 DDE（时滞微分方程）方程组。此示例最初由 Jackiewicz [1] 提出。

方程是：

$$y'(t) = 2 \cos(2t)y\left(\frac{t}{2}\right)^{2 \cos(t)} + \log\left(y\left(\frac{t}{2}\right)\right) - \log(2 \cos(t)) - \sin(t).$$

此方程是初始值 DDE，因为在 t_0 处时滞为零。因此，不需要历史解来计算解，只需要初始值：

$$y(0) = 1,$$

$$y'(0) = s.$$

s 是 $2 + \log(s) - \log(2) = 0$ 的解。满足此方程的 s 的值是 $s_1 = 2$ 和 $s_2 = 0.4063757399599599$ 。

由于方程中的时滞存在于 y' 项中，因此该方程称为中立型 DDE。

要在 MATLAB 中求解此方程，您需要先编写方程和时滞的代码，然后调用时滞微分方程求解器 **ddensd**，后者是中立型方程的求解器。您可以将所需的函数作为局部函数包含在文件末尾（如本处所示），或者将它们作为单独的文件保存在 MATLAB 路径上的目录中。

编写时滞代码

首先，编写一个匿名函数来定义方程中的迟滞。由于 y 和 y' 都有 $\frac{t}{2}$ 形式的迟滞，因此只需要一个函数定义。此时滞函数后来传递给求解器两次，一次表示 y 的时滞，一次表示 y' 的时滞。

```
delay = @(t,y) t/2;
```

编写方程代码

现在，创建一个函数来编写方程代码。此函数应具有签名 **yp = ddefun(t,y,ydel,ypdel)**，其中：

- t 是时间（自变量）。
- y 是解（因变量）。
- $ydel$ 包含 y 的时滞。
- $ypdel$ 包含 $y' = \frac{dy}{dt}$ 的时滞。

求解器会自动将这些输入传递给该函数，但是变量名称决定如何编写方程代码。在这种情况下：

- $ydel \rightarrow y\left(\frac{t}{2}\right)$
- $ypdel \rightarrow y'\left(\frac{t}{2}\right)$

```
function yp = ddefun(t,y,ydel,ypdel)
    yp = 2*cos(2*t)*ydel^(2*cos(t)) + log(ypdel) - log(2*cos(t)) - sin(t);
end
```

注意：所有函数都作为局部函数包含在示例的末尾。

求解方程

最后，定义积分区间 $[t_0 \ t_f]$ 和初始值，然后使用 `ddensd` 求解器求解 DDE。通过在第四个输入参数的元胞数组中指定初始值，将初始值传递给求解器。

```
tspan = [0 0.1];
y0 = 1;
s1 = 2;
sol1 = ddensd(@ddefun, delay, delay, {y0,s1}, tspan);
```

第二次求解方程，这次使用 s 的备选值作为初始条件。

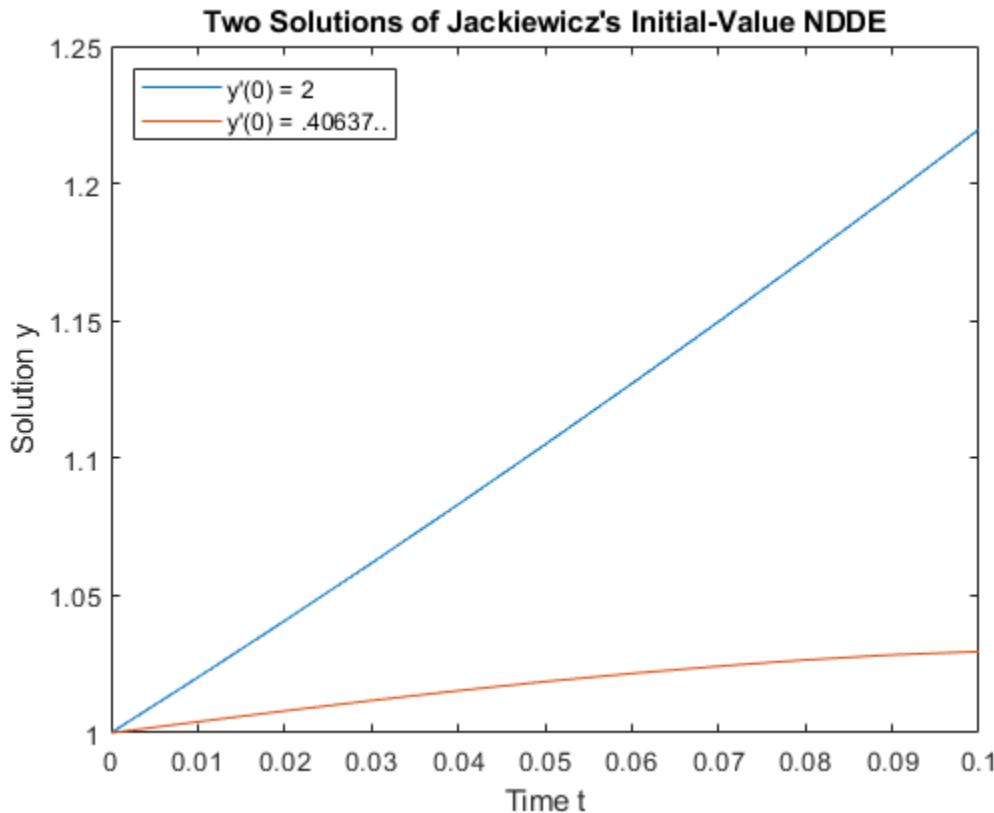
```
s2 = 0.4063757399599599;
sol2 = ddensd(@ddefun, delay, delay, {y0,s2}, tspan);
```

对解进行绘图

解结构体 `sol1` 和 `sol2` 具有字段 `x` 和 `y`，这些字段包含求解器在这些时间点所用的内部时间步和对应的解。但是，您可以使用 `deval` 计算在特定点的解。

绘制两个解以比较结果。

```
plot(sol1.x,sol1.y,sol2.x,sol2.y);
legend('y''(0) = 2','y''(0) = .40637..','Location','NorthWest');
xlabel('Time t');
ylabel('Solution y');
title('Two Solutions of Jackiewicz''s Initial-Value NDDE');
```



局部函数

此处列出了 DDE 求解器 **ddensd** 为计算解而调用的局部辅助函数。您也可以将这些函数作为它们自己的文件保存在 MATLAB 路径上的目录中。

```
function yp = ddefun(t,y,ydel,ypdel)
    yp = 2*cos(2*t)*ydel^(2*cos(t)) + log(ypdel) - log(2*cos(t)) - sin(t);
end
```

参考

[1] Jackiewicz, Z. "One step Methods of any Order for Neutral Functional Differential Equations." SIAM Journal on Numerical Analysis. Vol. 21, Number 3. 1984. pp. 486–511.

另请参阅

[dde23](#) | [ddensd](#) | [ddesd](#) | [deval](#)

详细信息

- “解算时滞微分方程” (第 14-2 页)
- “中立型 DDE” (第 14-17 页)

数值积分

- “计算弧线长度的积分” (第 15-2 页)
- “复曲线积分” (第 15-3 页)
- “积分域内部的奇点” (第 15-5 页)
- “多项式积分的解析解” (第 15-7 页)
- “数值数据的积分” (第 15-8 页)
- “计算表面的切平面” (第 15-12 页)

计算弧线长度的积分

此示例说明了如何参数化曲线以及使用 **integral** 计算弧线长度。

将曲线视为带有参数的方程

$$x(t) = \sin(2t), \quad y(t) = \cos(t), \quad z(t) = t,$$

其中 $t \in [0, 3\pi]$ 。

创建此曲线的三维绘图。

```
t = 0:0.1:3*pi;
plot3(sin(2*t),cos(t),t)
```

弧线长度公式表明曲线的长度是参数化方程的导数范数的积分。

$$\int_0^{3\pi} \sqrt{4\cos^2(2t) + \sin^2(t) + 1} \, dt.$$

将被积函数定义为匿名函数。

```
f = @(t) sqrt(4*cos(2*t).^2 + sin(t).^2 + 1);
```

通过调用 **integral** 对此函数进行积分计算。

```
len = integral(f,0,3*pi)
```

```
len =
17.2220
```

此曲线的长度大约为 17.2。

另请参阅

integral

详细信息

- “创建函数句柄”
- “积分域内部的奇点”（第 15-5 页）
- “数值数据的积分”（第 15-8 页）

复曲线积分

此示例说明如何使用 `integral` 函数的 'Waypoints' 选项计算复曲线积分。在 MATLAB® 中，可以使用 'Waypoints' 选项定义直线路径序列，从第一个积分限值到第一个路径点，从第一个路径点到第二个路径点，依此类推，直到从最后一个路径点到第二个积分限值。

将被积函数定义为匿名函数

对以下方程求积分

$$\oint_C \frac{e^z}{z} dz$$

其中 C 是一条闭围线，围绕原点处的 e^z/z 简单极点。

将被积函数定义为匿名函数。

```
fun = @(z) exp(z)./z;
```

不使用路径点求积分

可以用参数化计算复值函数的围线积分。在一般情况下，指定一条围线，然后将其微分并用于参数化原被积函数。在这种情况下，将围线作为单位圆，但在所有情况下，其结果与所选围线无关。

```
g = @(theta) cos(theta) + 1i*sin(theta);
gprime = @(theta) -sin(theta) + 1i*cos(theta);
q1 = integral(@(t) fun(g(t)).*gprime(t),0,2*pi)

q1 = -0.0000 + 6.2832i
```

这种参数化方法虽然可靠，但难以计算和费时，因为必须先计算导数，然后才能积分。即使是简单函数，也需要写几行代码才能获得正确的结果。由于围绕极点（在本例中为原点）的任何闭围线都有相同的结果，因此可以使用 `integral` 的 'Waypoints' 选项构建一个围绕极点的方形或三角形路径。

对不包含极点的围线求积分

如果路径点向量积分或元素限值为复数，则 `integral` 会在复平面中针对直线路径序列求积分。围线周围的自然方向为逆时针；指定顺时针围线类似于乘以 `-1`。以这种方式指定围线使其包含一个单函数奇点。如果指定一条不包含极点的围线，则柯西积分定理可保证闭积分环的值是零。

为此，应对远离原点的方围线周围的 `fun` 求积分。使用相等的积分限值形成一个闭围线。

```
C = [2+i 2+2i 1+2i];
q = integral(fun,1+i,1+i,'Waypoints',C)

q = -3.3307e-16 + 6.6613e-16i
```

其结果数量级为 `eps`，实际上为零。

对内部包含极点的围线求积分

指定一个完全在原点包含极点的方围线，然后求积分。

```
C = [1+i -1+i -1-i 1-i];
q2 = integral(fun,1,1,'Waypoints',C)

q2 = -0.0000 + 6.2832i
```

这个结果与 q1 的上述计算相符，但使用的代码简单得多。

这个问题的确切答案是 $2\pi i$ 。

2*pi*i

ans = 0.0000 + 6.2832i

另请参阅

[integral](#)

详细信息

- “[创建函数句柄](#)”
- “[积分域内部的奇点](#)”（第 15-5 页）
- “[数值数据的积分](#)”（第 15-8 页）

积分域内部的奇点

本示例显示如何拆分积分域以将奇点放在边界上。

将被积函数定义为匿名函数

复值积分的被积函数

$$\int_{-1}^1 \int_{-1}^1 \frac{1}{\sqrt{x+y}} dx dy$$

在 $x = y = 0$ 时有一个奇点，并通常是 $y = -x$ 线上的奇异值。

将该被积函数定义为匿名函数。

```
fun = @(x,y) ((x+y).^( -1/2));
```

对方形求积分

对由 $-1 \leq x \leq 1$ 和 $-1 \leq y \leq 1$ 指定的方域中的 **fun** 求积分。

```
format long
q = integral2(fun,-1,1,-1,1)
```

Warning: Non-finite result. The integration was unsuccessful. Singularity likely.

```
q =
NaN +      NaNi
```

如果积分区内部有奇异值，则积分不能收敛并返回一个警告。

将积分域拆分为两个三角形

可以通过将积分域拆分为互补区并将这些较小的积分加在一起重新定义积分。可将奇点放在域边界上来避免积分错误和警告。在此例中，可将方积分区域沿着奇异线 $y = -x$ 拆分成两个三角形并将结果相加。

```
q1 = integral2(fun,-1,1,-1,@(x)-x);
q2 = integral2(fun,-1,1,@(x)-x,1);
q = q1 + q2

q =
3.771236166328258 - 3.771236166328256i
```

奇异值在边界上时可继续求积分。

这个积分的精确值是

$$\frac{8\sqrt{2}}{3}(1 - i)$$

```
8/3*sqrt(2)*(1-i)
```

```
ans =  
3.771236166328253 - 3.771236166328253i
```

另请参阅

[integral](#) | [integral2](#) | [integral3](#)

详细信息

- “[创建函数句柄](#)”
- “[复曲线积分](#)”（第 15-3 页）
- “[数值数据的积分](#)”（第 15-8 页）

多项式积分的解析解

本示例显示如何使用 **polyint** 函数对多项式求解析积分。使用此函数来计算多项式的不定积分。

定义问题

考虑实数不定积分，

$$\int (4x^5 - 2x^3 + x + 4) dx$$

被积函数是多项式，解析解是

$$\frac{2}{3}x^6 - \frac{1}{2}x^4 + \frac{1}{2}x^2 + 4x + k$$

其中 k 是积分常量。由于没有指定积分限值，**integral** 函数族不太适合求解这个问题。

用向量表示多项式

创建一个向量，其元素代表各 x 降幂的系数。

```
p = [4 0 -2 0 1 4];
```

对多项式求解析积分

使用 **polyint** 函数求多项式的解析积分。指定第二输入参数的积分常量。

```
k = 2;
I = polyint(p,k)
```

```
I = 1×7
```

```
0.6667      0   -0.5000      0    0.5000    4.0000   2.0000
```

输出是一个 x 降幂系数向量。这一结果与上述解析解相匹配，但有积分常量 $k = 2$ 。

另请参阅

polyint | polyval

详细信息

- “积分域内部的奇点”（第 15-5 页）
- “数值数据的积分”（第 15-8 页）

数值数据的积分

此示例显示如何对一组离散速度数据进行数值积分以逼近行驶距离。integral 族仅接受函数句柄输入，所以这些函数不能用于离散数据集。当函数表达式不能用于积分时，使用 trapz 或 cumtrapz。

查看速度数据

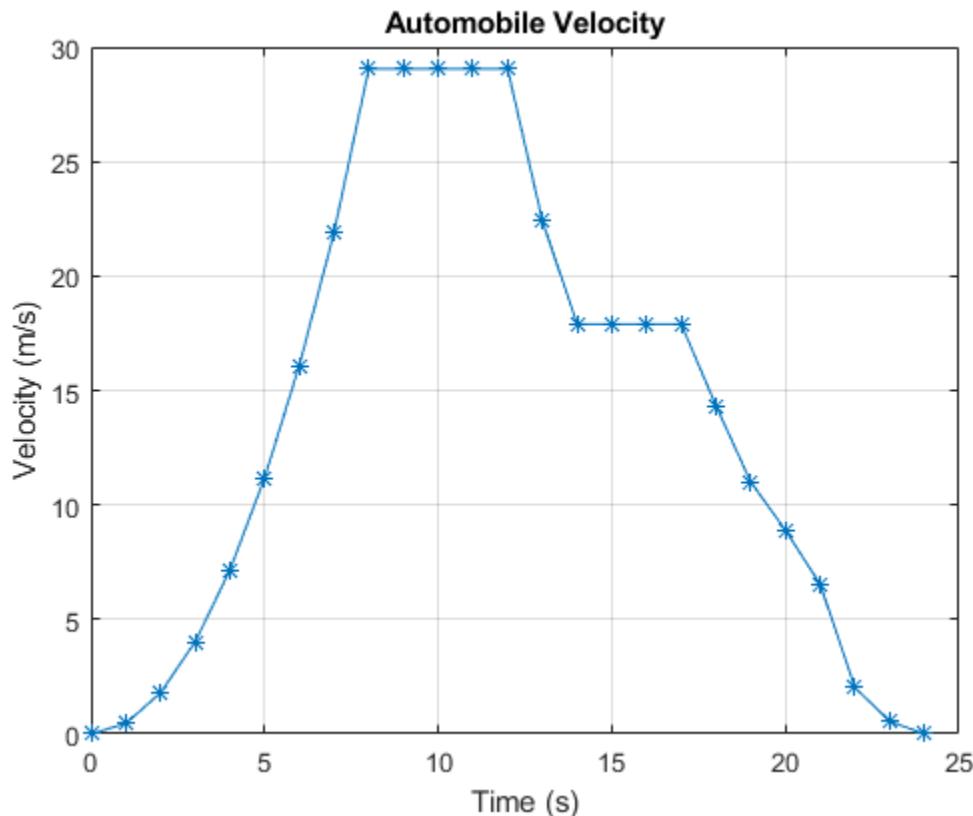
考虑以下速度数据和相应的时间数据。

```
vel = [0.45 1.79 4.02 7.15 11.18 16.09 21.90 29.05 29.05 ...
29.05 29.05 29.05 22.42 17.9 17.9 17.9 17.9 14.34 11.01 ...
8.9 6.54 2.03 0.55 0];
time = 0:24;
```

这些数据代表汽车的速度（米/秒），间隔为 1 秒，时间超过 24 秒。

绘制速度数据点并将各点用直线连接。

```
figure
plot(time,vel,'-*')
grid on
title('Automobile Velocity')
xlabel('Time (s)')
ylabel('Velocity (m/s)')
```



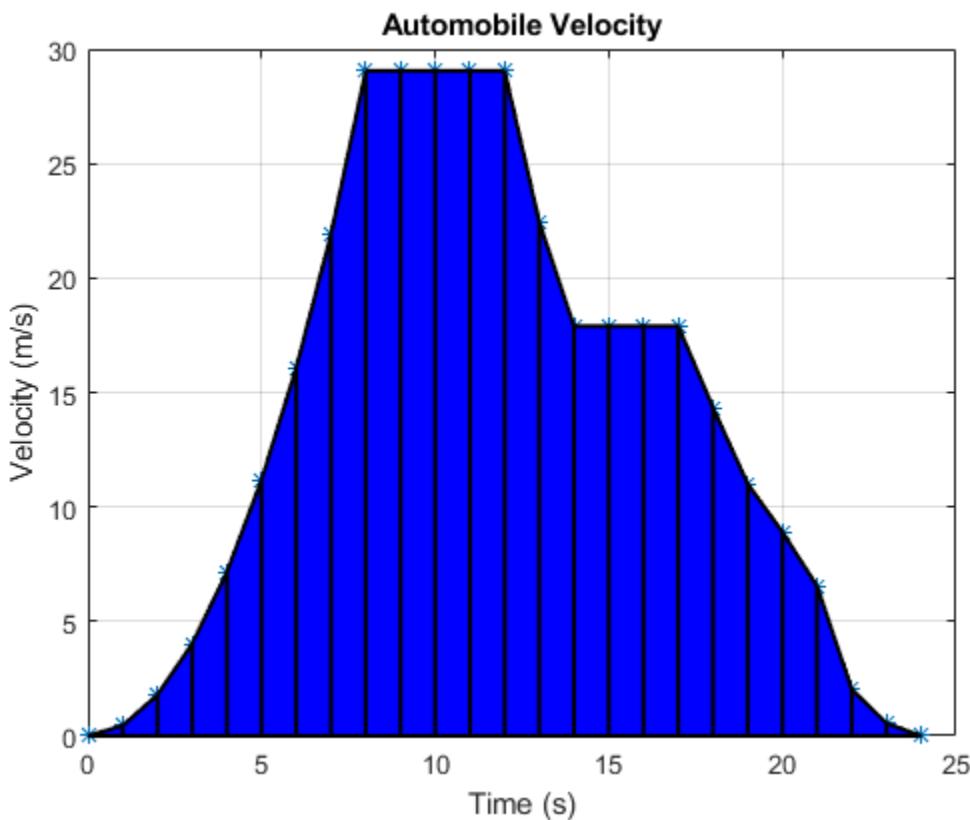
斜率在加速时为正，恒速时为零，减速时为负。在 $t = 0$ 的时间点，车辆处于静止，速度为 $vel(1) = 0$ 米/秒。然后车辆以 $vel(9) = 29.05$ 米/秒的速度加速，并在 $t = 8$ 秒内达到最大速度，并保持这种速度 4 秒的

时间。然后车辆在 3 秒之内减速到 $\text{vel}(14) = 17.9$ 米/秒并最终静止。由于这个速度曲线有多处不连续，因此不能用单一连续函数来描述。

计算总行驶距离

`trapz` 使用数据点进行离散积分以创建梯形，所以它非常适合处理不连续的数据集。这种方法假设在数据点之间为线性行为，当数据点之间的行为是非线性时，精度可能会降低。为了说明这一点，可将数据点作为顶点在图表上画出梯形。

```
xverts = [time(1:end-1); time(1:end-1); time(2:end); time(2:end)];
yverts = [zeros(1,24); vel(1:end-1); vel(2:end); zeros(1,24)];
p = patch(xverts,yverts,'b','LineWidth',1.5);
```



`trapz` 可通过将区域分解成梯形来计算离散数据集下的面积。然后，函数将每个梯形面积累加来计算总面积。

通过使用 `trapz` 求速度数据积分来计算汽车的总行驶距离（对应的着色区域）。默认情况下，如果使用语法 `trapz(Y)`，则假定点之间的间距为 1。但是，您可以使用语法 `trapz(X,Y)` 指定不同的均匀或非均匀间距 X。在这种情况下，`time` 向量中读数之间的间距是 1，因此可以使用默认间距。

```
distance = trapz(vel)
```

```
distance = 345.2200
```

汽车在 $t = 24$ 秒内行驶的距离约为 345.22 米。

绘制累积行驶距离

`cumtrapz` 函数与 `trapz` 密切相关。`trapz` 仅返回最终的积分值，而 `cumtrapz` 还在向量中返回中间值。

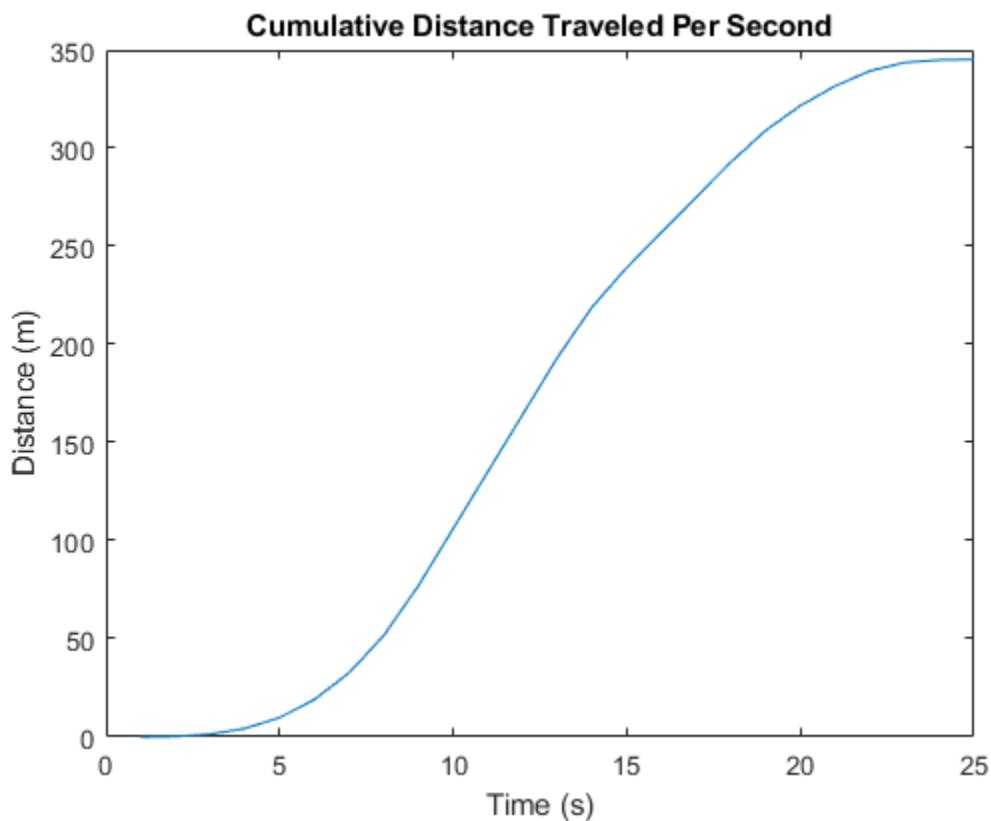
计算累积行驶距离并绘制结果。

```
cdistance = cumtrapz(vel);
T = table(time',cdistance','VariableNames',{'Time','CumulativeDistance'})
```

T=25×2 table
Time CumulativeDistance

0	0
1	0.225
2	1.345
3	4.25
4	9.835
5	19
6	32.635
7	51.63
8	77.105
9	106.15
10	135.2
11	164.25
12	193.31
13	219.04
14	239.2
15	257.1
:	

```
plot(cdistance)
title('Cumulative Distance Traveled Per Second')
xlabel('Time (s)')
ylabel('Distance (m)')
```



另请参阅

[cumtrapz](#) | [integral](#) | [trapz](#)

详细信息

- “积分域内部的奇点” (第 15-5 页)
- “多项式积分的解析解” (第 15-7 页)

计算表面的切平面

此示例说明如何按有限差分逼近函数梯度。然后说明如何通过使用这些逼近的梯度，绘制平面上某个点的切平面。

使用函数句柄创建函数 $f(x, y) = x^2 + y^2$ 。

```
f = @(x,y) x.^2 + y.^2;
```

使用 **gradient** 函数，相对 x 和 y 逼近 $f(x, y)$ 的偏导数。选择与网格大小相同的有限差分长度。

```
[xx,yy] = meshgrid(-5:0.25:5);
[fx,fy] = gradient(f(xx,yy),0.25);
```

曲面上的点 $P = (x_0, y_0, f(x_0, y_0))$ 的切平面表示为

$$z = f(x_0, y_0) + \frac{\partial f(x_0, y_0)}{\partial x}(x - x_0) + \frac{\partial f(x_0, y_0)}{\partial y}(y - y_0).$$

\mathbf{fx} 和 \mathbf{fy} 矩阵是偏导数 $\frac{\partial f}{\partial x}$ 和 $\frac{\partial f}{\partial y}$ 的近似值。此示例中的相关点（即切平面与函数平面的接合点）为 $(x_0, y_0) = (1, 2)$ 。此相关点位置的函数值为 $f(1, 2) = 5$ 。

为逼近切平面 z ，您需要求取相关点的导数值。获取该点的索引，并求取该位置的近似导数。

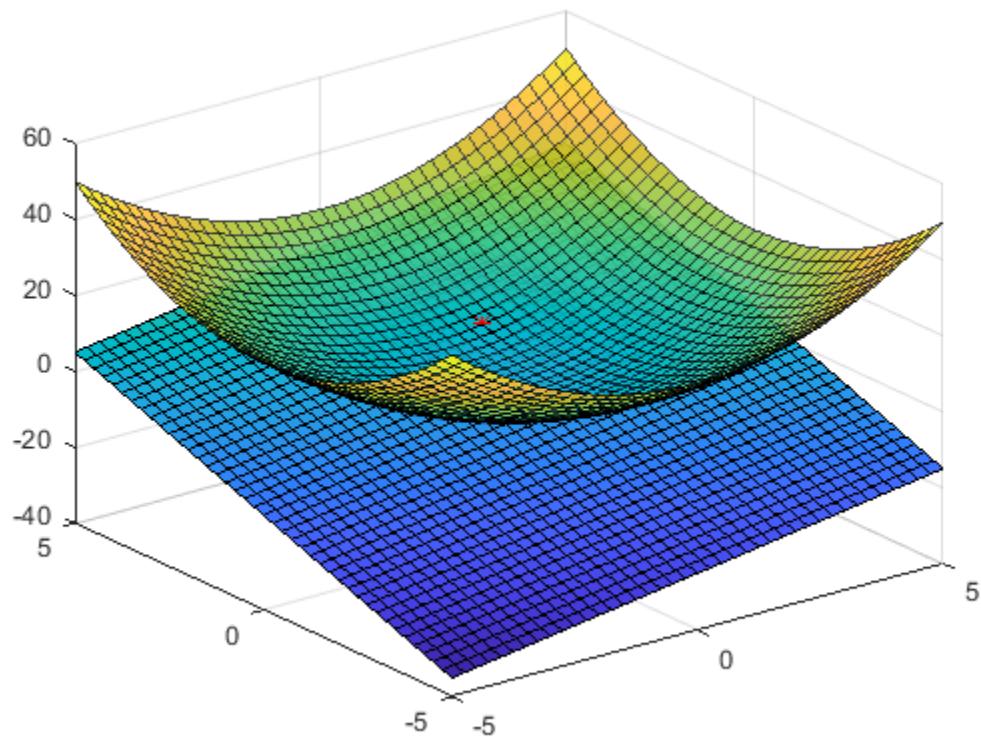
```
x0 = 1;
y0 = 2;
t = (xx == x0) & (yy == y0);
indt = find(t);
fx0 = fx(indt);
fy0 = fy(indt);
```

使用切平面 z 的方程创建函数句柄。

```
z = @(x,y) f(x0,y0) + fx0*(x-x0) + fy0*(y-y0);
```

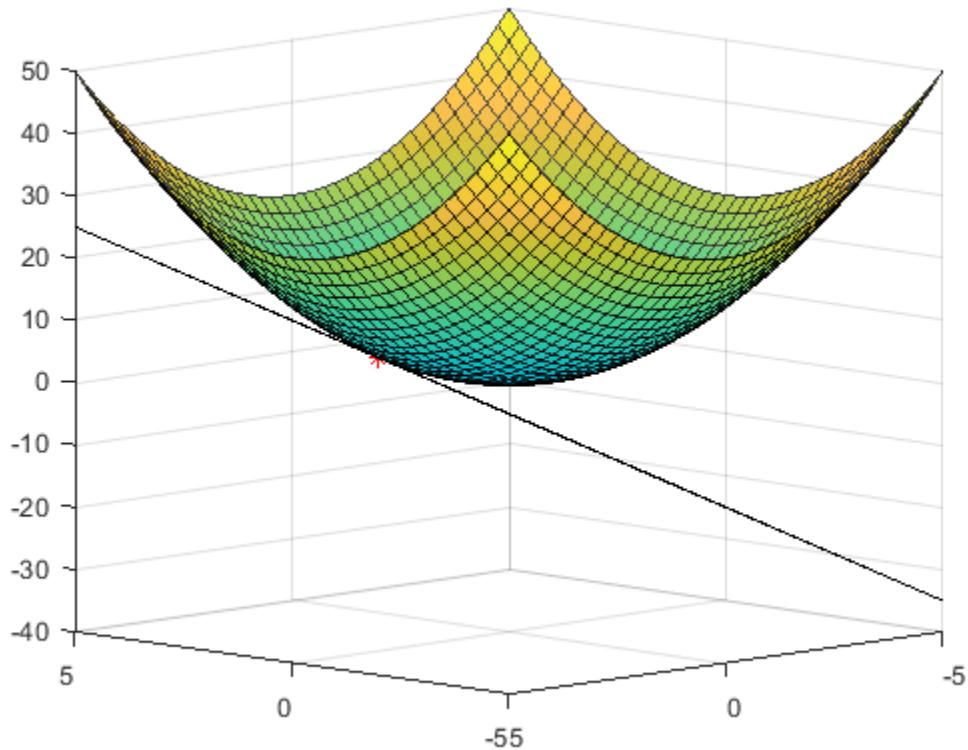
绘制原始函数 $f(x, y)$ 、点 P ，以及在 P 位置与函数相切的平面 z 的片段。

```
surf(xx,yy,f(xx,yy),'EdgeAlpha',0.7,'FaceAlpha',0.9)
hold on
surf(xx,yy,z(xx,yy))
plot3(1,2,f(1,2),'r*')
```



查看侧剖图。

`view(-135,9)`



另请参阅

`gradient`

详细信息

- “[创建函数句柄](#)”

傅里叶变换

- “傅里叶变换” (第 16-2 页)
- “基本频谱分析” (第 16-8 页)
- “使用 FFT 进行多项式插值” (第 16-14 页)
- “二维傅里叶变换” (第 16-17 页)
- “使用 FFT 进行频谱分析” (第 16-21 页)
- “从正弦波转换为方波” (第 16-24 页)
- “使用 FFT 分析周期性数据” (第 16-29 页)

傅里叶变换

傅里叶变换是将按时间或空间采样的信号与按频率采样的相同信号进行关联的数学公式。在信号处理中，傅里叶变换可以揭示信号的重要特征（即其频率分量）。

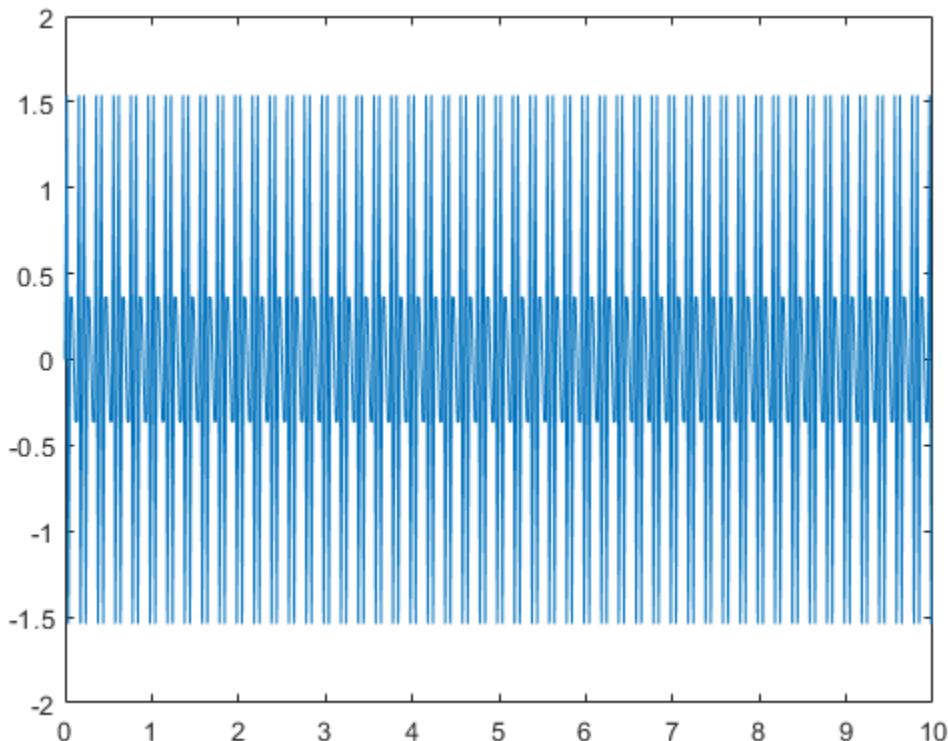
对于包含 n 个均匀采样点的向量 x ，其傅里叶变换定义为

$$y_{k+1} = \sum_{j=0}^{n-1} \omega^{jk} x_{j+1}.$$

$\omega = e^{-2\pi i/n}$ 是 n 个复单位根之一，其中 i 是虚数单位。对于 x 和 y ，索引 j 和 k 的范围为 0 到 $n - 1$ 。

MATLAB® 中的 `fft` 函数使用快速傅里叶变换算法来计算数据的傅里叶变换。以正弦信号 x 为例，该信号是时间 t 的函数，频率分量为 15 Hz 和 20 Hz。使用在 10 秒周期内以 $\frac{1}{50}$ 秒为增量进行采样的时间向量。

```
t = 0:1/50:10-1/50;
x = sin(2*pi*15*t) + sin(2*pi*20*t);
plot(t,x)
```

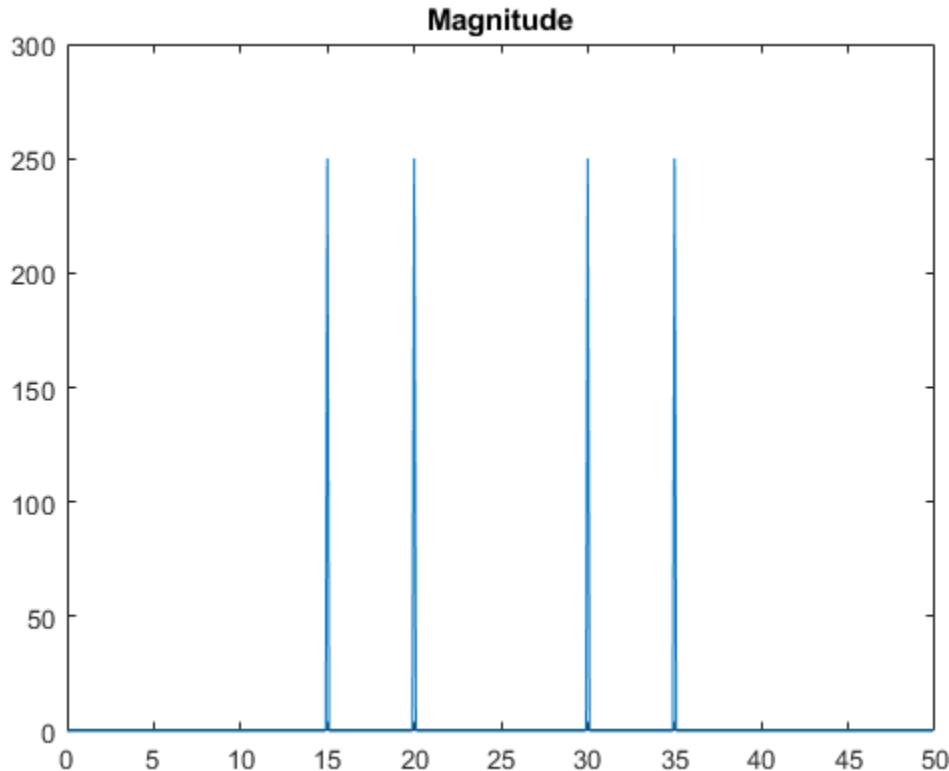


计算信号的傅里叶变换，并在频率空间创建对应于信号采样的向量 f 。

```
y = fft(x);
f = (0:length(y)-1)*50/length(y);
```

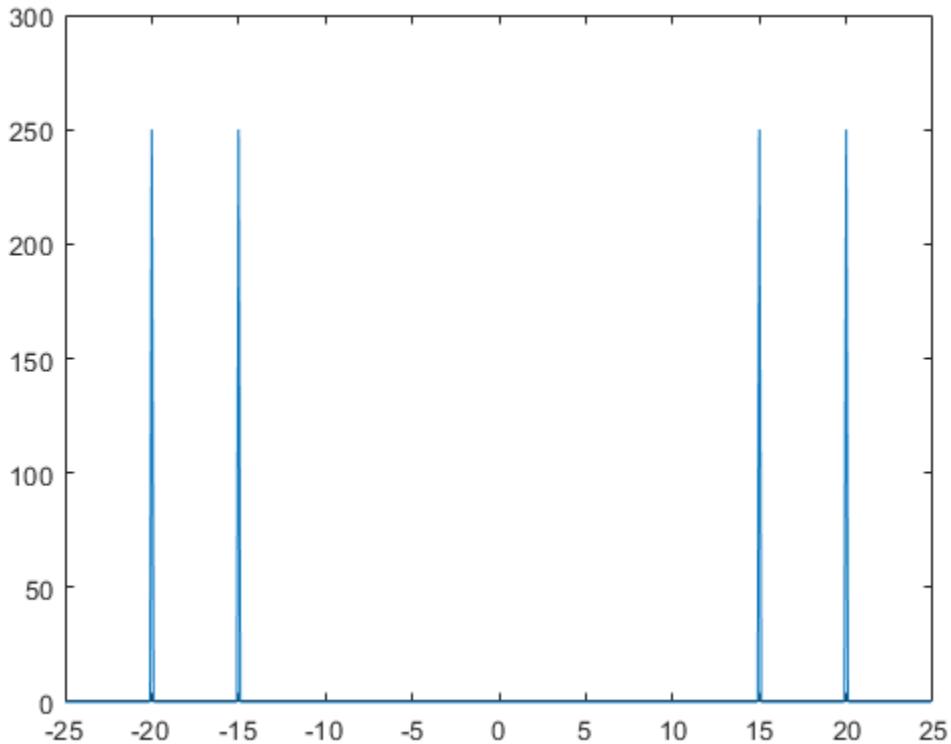
以频率函数形式绘制信号幅值时，幅值尖峰对应于信号的 15 Hz 和 20 Hz 频率分量。

```
plot(f,abs(y))
title('Magnitude')
```



该变换还会生成尖峰的镜像副本，该副本对应于信号的负频率。为了更好地以可视化方式呈现周期性，您可以使用 `fftshift` 函数对变换执行以零为中心的循环平移。

```
n = length(x);
fshift = (-n/2:n/2-1)*(50/n);
yshift = fftshift(y);
plot(fshift,abs(yshift))
```



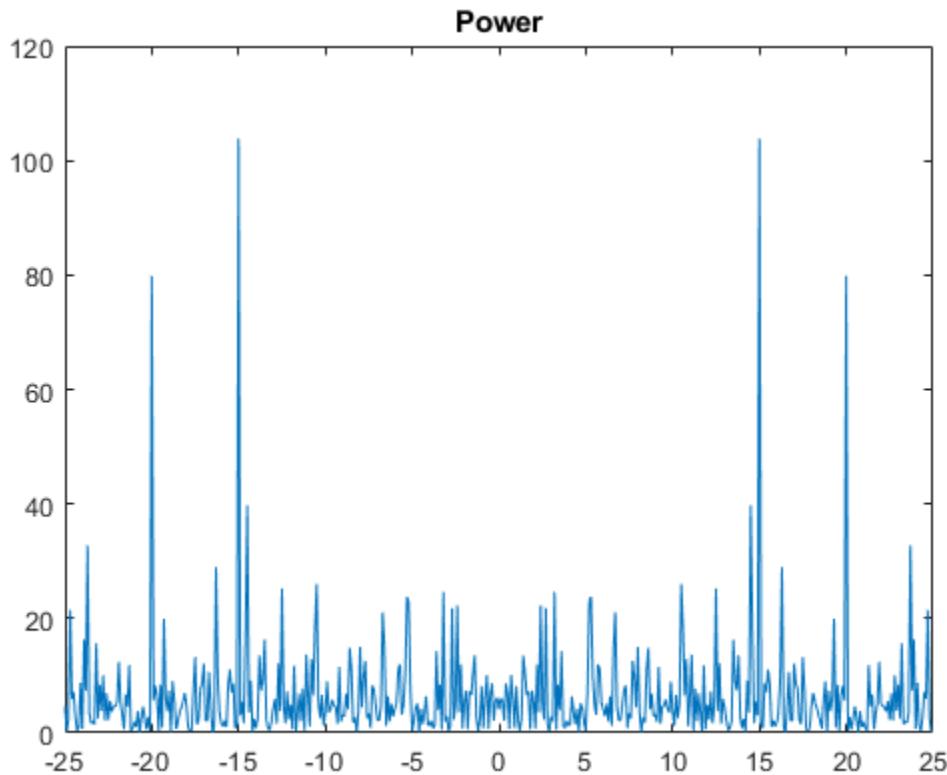
含噪信号

在科学应用中，信号经常遭到随机噪声破坏，掩盖其频率分量。傅里叶变换可以清除随机噪声并显现频率。例如，通过在原始信号 `x` 中注入高斯噪声，创建一个新信号 `xnoise`。

```
xnoise = x + 2.5*gallery('normaldata',size(t),4);
```

频率函数形式的信号功率是信号处理中的一种常用度量。功率是信号的傅里叶变换按频率样本数进行归一化后的平方幅值。计算并绘制以零频率为中心的含噪信号的功率谱。尽管存在噪声，您仍可以根据功率中的尖峰辨识出信号的频率。

```
ynoise = fft(xnoise);
ynoiseshift = fftshift(ynoise);
power = abs(ynoiseshift).^2/n;
plot(fshift,power)
title('Power')
```



计算效率

直接使用傅里叶变换公式分别计算 y 的 n 个元素需要 n^2 数量级的浮点运算。使用快速傅里叶变换算法，则只需要 $n \log n$ 数量级的运算。在处理包含成百上千万个数据点的数据时，这一计算效率会带来很大的优势。在 n 为 2 的幂时，许多专门的快速傅里叶变换实现可进一步提高效率。

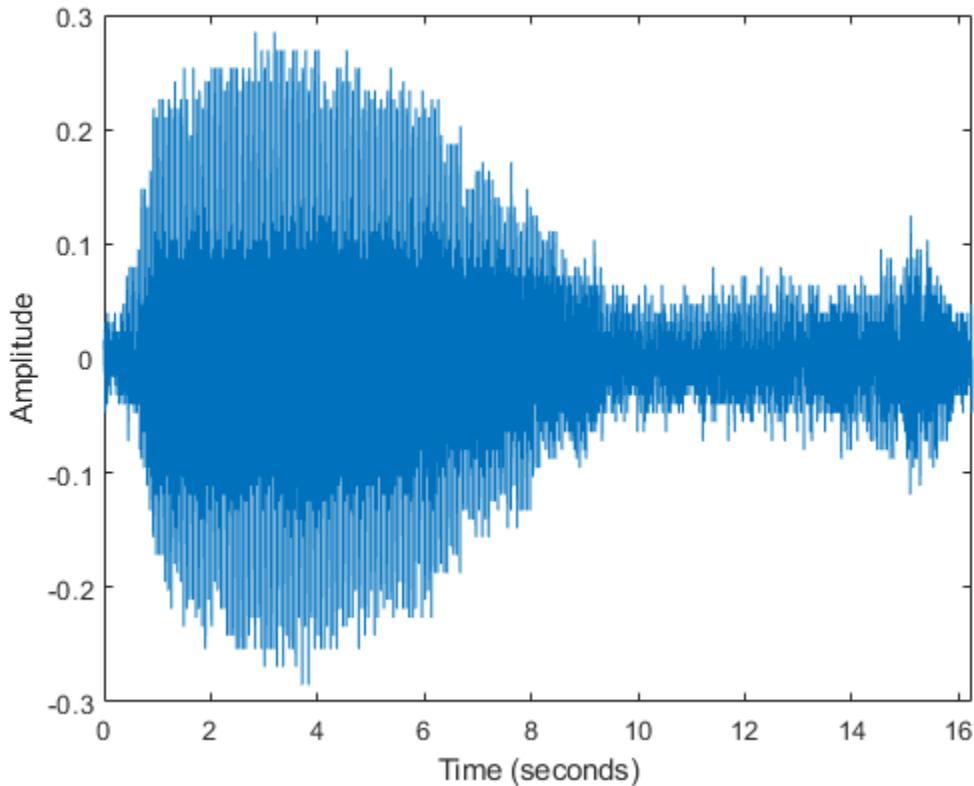
以加利福尼亚海岸的水下麦克风所收集的音频数据为例。在康奈尔大学生物声学研究项目维护的库中可以找到这些数据。载入包含太平洋蓝鲸鸣声的文件 `bluewhale.au`，并对其中一部分数据进行格式化。可使用命令 `sound(x,fs)` 来收听完整的音频文件。

```

whaleFile = 'bluewhale.au';
[x,fs] = audioread(whaleFile);
whaleMoan = x(2.45e4:3.10e4);
t = 10*(0:1/fs:(length(whaleMoan)-1)/fs);

plot(t,whaleMoan)
xlabel('Time (seconds)')
ylabel('Amplitude')
xlim([0 t(end)])

```

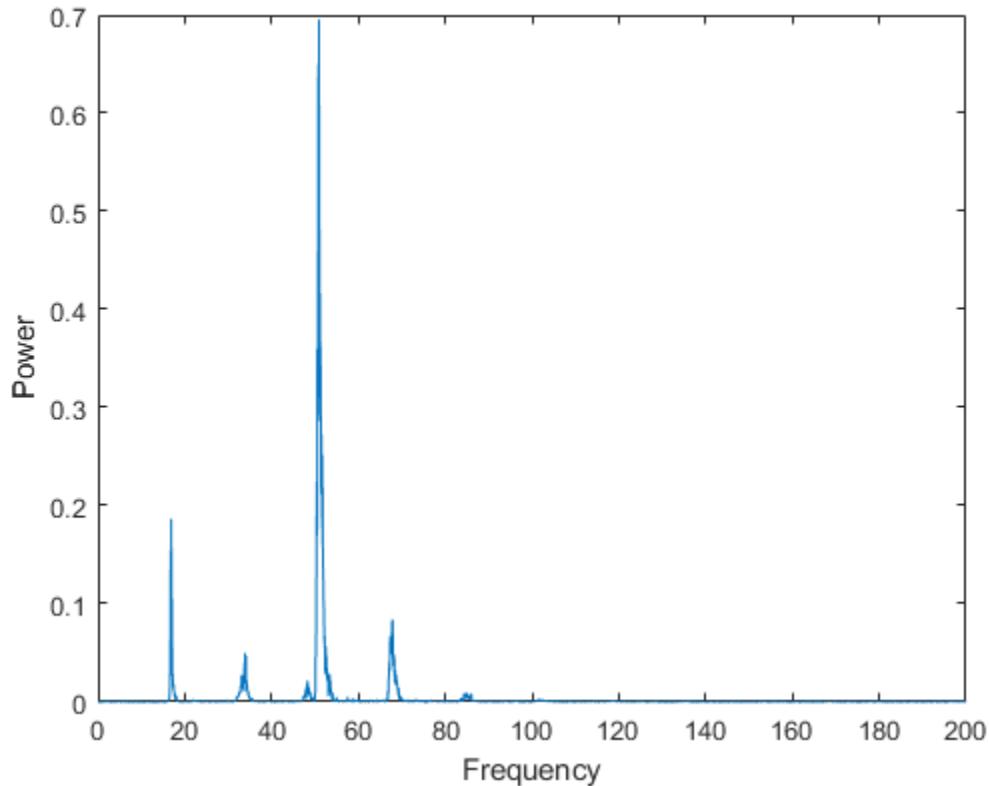


指定新的信号长度，该长度是大于原始长度的最邻近的 2 的幂。然后使用 `fft` 和新的信号长度计算傅里叶变换。`fft` 会自动用零填充数据，以增加样本大小。此填充操作可以大幅提高变换计算的速度，对于具有较大质因数的样本大小更是如此。

```
m = length(whaleMoan);
n = pow2(nextpow2(m));
y = fft(whaleMoan,n);
```

绘制信号的功率谱。绘图指示，呻吟音包含约 17 Hz 的基本频率和一系列谐波（其中强调了第二个谐波）。

```
f = (0:n-1)*(fs/n)/10; % frequency vector
power = abs(y).^2/n; % power spectrum
plot(f(1:floor(n/2)),power(1:floor(n/2)))
xlabel('Frequency')
ylabel('Power')
```



另请参阅

[ifft](#) | [fft2](#) | [fftn](#) | [fftw](#) | [fft](#) | [fftshift](#) | [nextpow2](#)

相关示例

- “二维傅里叶变换” (第 16-17 页)

基本频谱分析

傅里叶变换是用于对时域信号执行频率和功率谱分析的工具。

频谱分析数量

频谱分析研究非均匀采样的离散数据中包含的频谱。傅里叶变换是通过在频率空间表示基于时间或空间的信号来揭示该信号的频率分量的工具。下表列出了用于描述和解释信号属性的常用量。要了解有关傅里叶变换的更多信息，请参阅“傅里叶变换”（第 16-2 页）。

数量	说明
<code>x</code>	采样的数据
<code>n = length(x)</code>	样本数量
<code>fs</code>	采样频率（每单位时间或空间的样本数）
<code>dt = 1/fs</code>	每样本的时间或空间增量
<code>t = (0:n-1)/fs</code>	数据的时间或空间范围
<code>y = fft(x)</code>	数据的离散傅里叶变换 (DFT)
<code>abs(y)</code>	DFT 的振幅
<code>(abs(y).^2)/n</code>	DFT 的幂
<code>fs/n</code>	频率增量
<code>f = (0:n-1)*(fs/n)</code>	频率范围
<code>fs/2</code>	Nyquist 频率（频率范围的中点）

含噪信号

傅里叶变换可以计算被随机噪声破坏的信号的频率分量。

创建具有 15 Hz 和 40 Hz 分量频率的信号，并插入随机高斯噪声。

```
fs = 100; % sample frequency (Hz)
t = 0:1/fs:10-1/fs; % 10 second span time vector
x = (1.3)*sin(2*pi*15*t) ... % 15 Hz component
+ (1.7)*sin(2*pi*40*(t-2)) ... % 40 Hz component
+ 2.5*gallery('normaldata',size(t),4); % Gaussian noise;
```

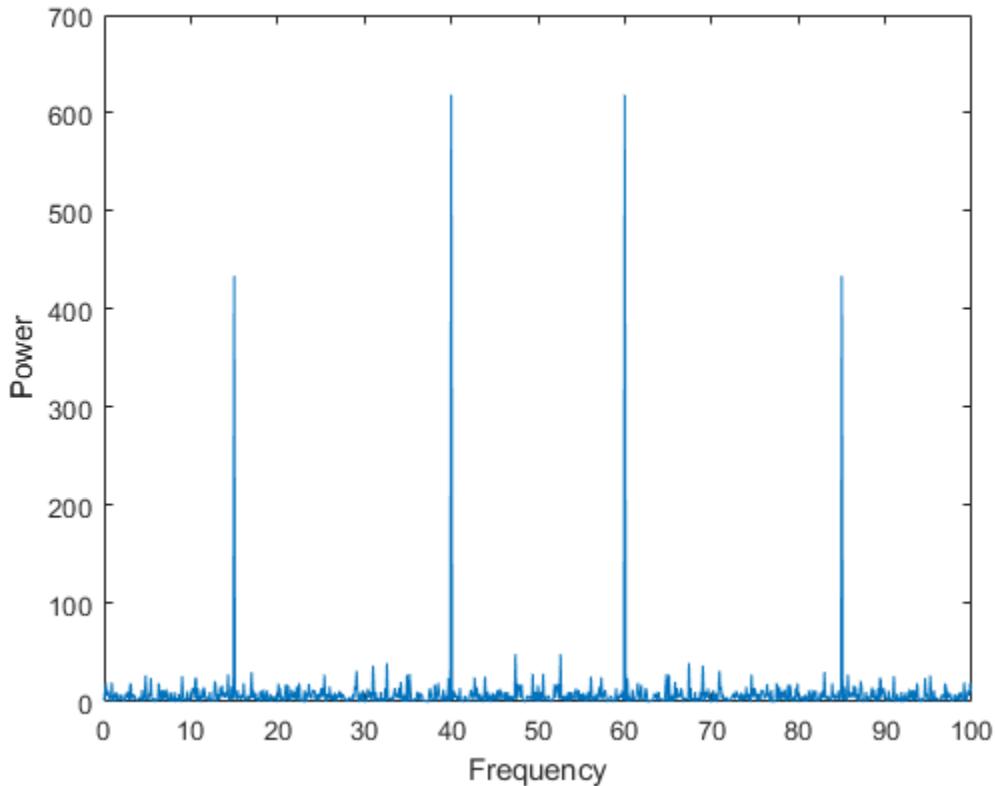
信号的傅里叶变换可确定其频率分量。在 MATLAB® 中，`fft` 函数使用快速傅里叶变换算法计算傅里叶变换。使用 `fft` 计算信号的离散傅里叶变换。

```
y = fft(x);
```

将功率谱绘制为频率的函数。尽管噪声在基于时间的空间内伪装成信号的频率分量，但傅里叶变换将其显现出为功率尖峰。

```
n = length(x); % number of samples
f = (0:n-1)*(fs/n); % frequency range
power = abs(y).^2/n; % power of the DFT

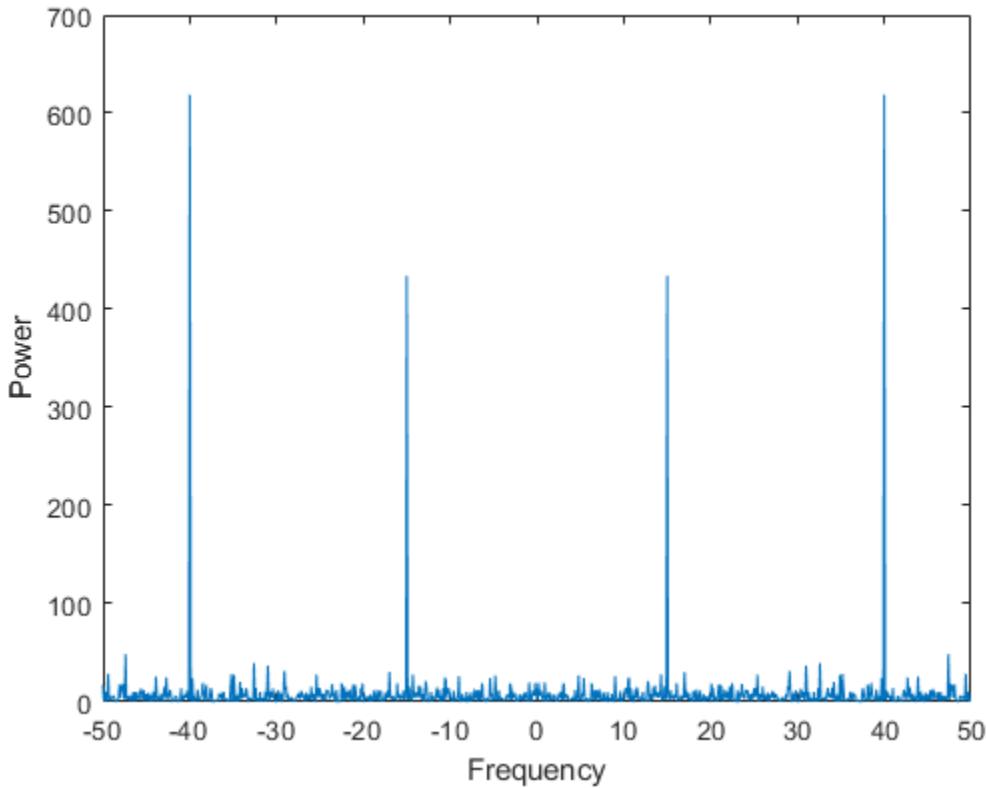
plot(f,power)
xlabel('Frequency')
ylabel('Power')
```



在许多应用中，查看以 0 频率为中心的功率谱更加方便，因为它能更好地显示信号的周期性。使用 `fftshift` 函数对 `y` 执行循环平移，并绘制以 0 为中心的功率。

```
y0 = fftshift(y);      % shift y values
f0 = (-n/2:n/2-1)*(fs/n); % 0-centered frequency range
power0 = abs(y0).^2/n;   % 0-centered power

plot(f0,power0)
xlabel('Frequency')
ylabel('Power')
```



音频信号

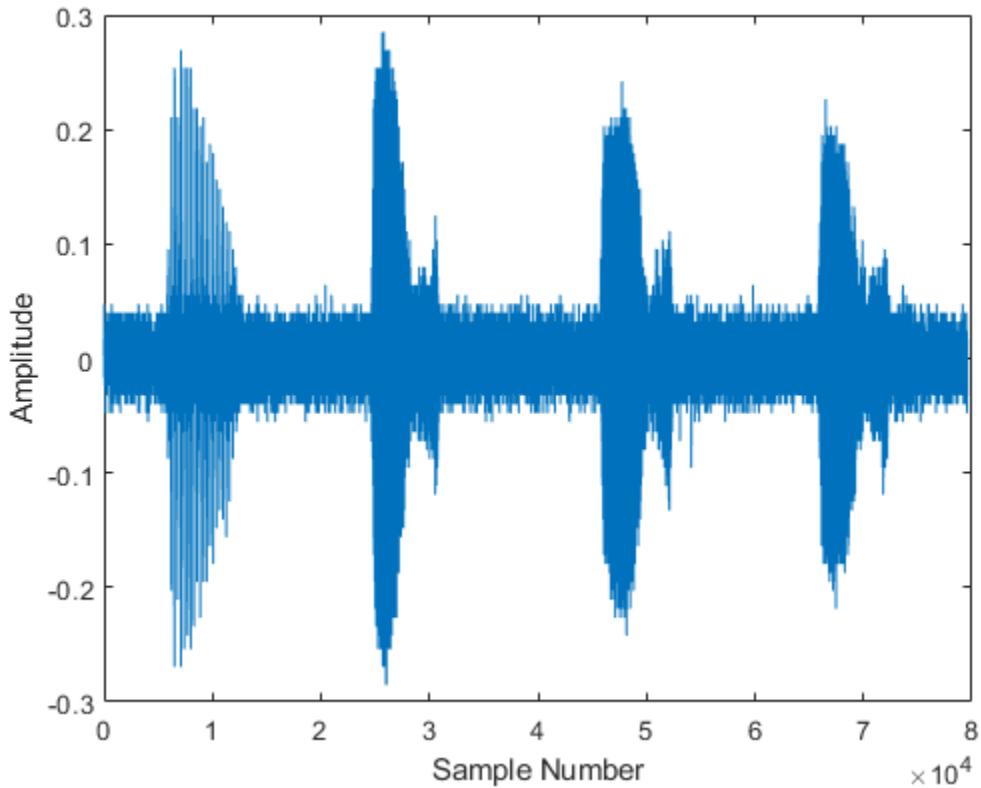
您可以使用傅里叶变换来分析音频数据的频谱。

文件 **bluewhale.au** 包含水下麦克风记录的加利福尼亚海岸的太平洋蓝鲸发声的音频数据。此文件来自于康奈尔大学生物声学研究项目保存的动物发声库。

由于蓝鲸的叫声频率如此之低，以至人类几乎听不到。数据中的时间标度压缩了 10 倍，以便提高音调并使叫声更清晰可闻。读取并绘制音频数据。可使用命令 **sound(x,fs)** 来收听音频。

```
whaleFile = 'bluewhale.au';
[x,fs] = audioread(whaleFile);

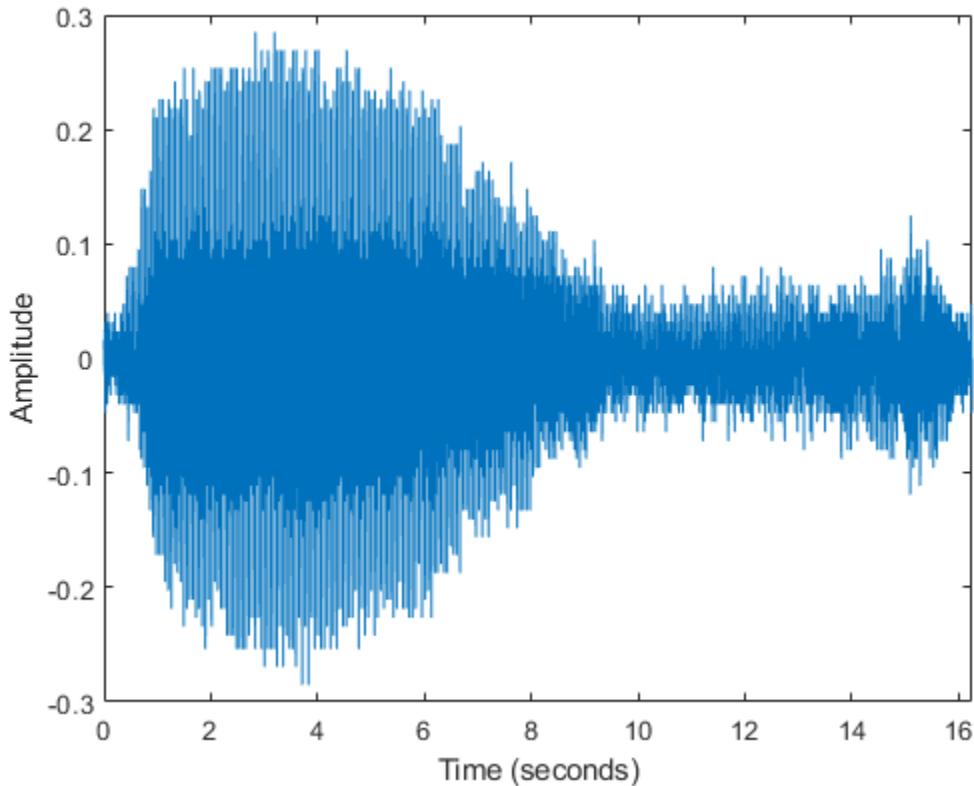
plot(x)
xlabel('Sample Number')
ylabel('Amplitude')
```



第一个声音为“颤音”，之后是三个“呻吟音”。本示例将分析单个呻吟音。指定大致包含第一个呻吟音的新数据，并校正时间数据以体现 10 部的加速。将截断的信号绘制为时间的函数。

```
moan = x(2.45e4:3.10e4);
t = 10*(0:1/fs:(length(moan)-1)/fs);

plot(t,moan)
xlabel('Time (seconds)')
ylabel('Amplitude')
xlim([0 t(end)])
```



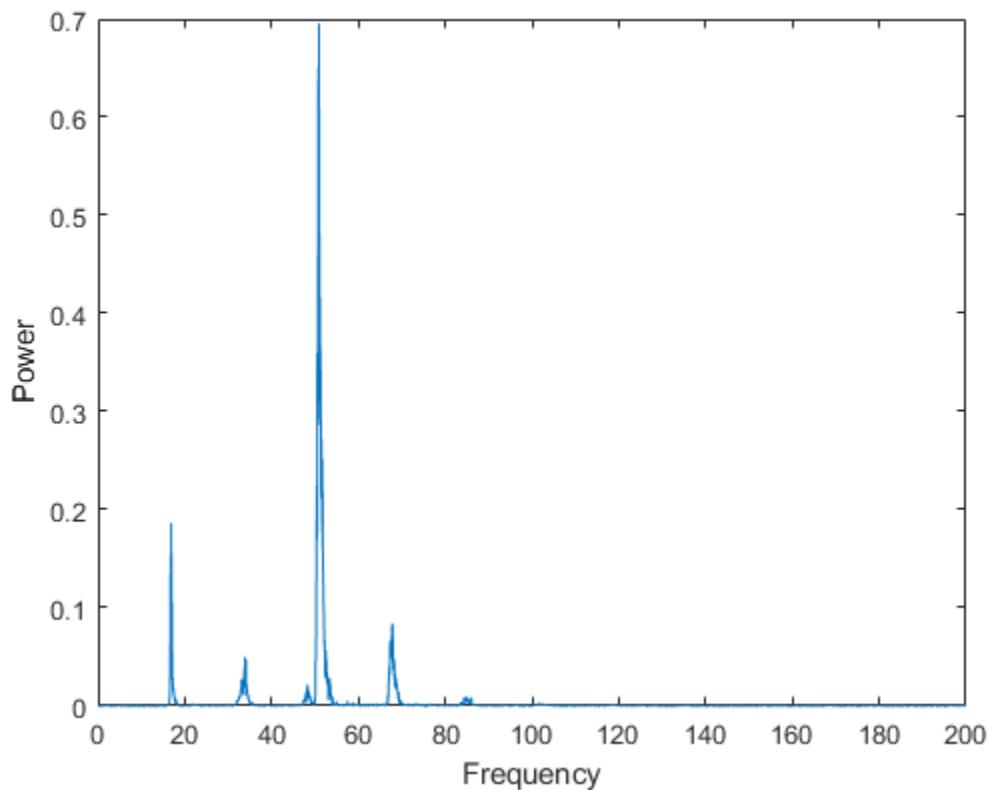
数据的傅里叶变换确定了音频信号的频率分量。在一些使用 `fft` 处理大量数据的应用中，通常需要调整输入，使样本数量为 2 的幂。这样可以大幅提高变换计算的速度，对于具有较大质因数的样本大小更是如此。指定新的信号长度 `n` (2 的幂)，并使用 `fft` 函数计算信号的离散傅里叶变换。`fft` 会自动使用零来填充原始数据，以增加样本大小。

```
m = length(moan); % original sample length
n = pow2(nextpow2(m)); % transform length
y = fft(moan,n); % DFT of signal
```

根据加速因子调整频率范围，并计算和绘制信号的功率谱。绘图指示，呻吟音包含约 17 Hz 的基本频率和一系列谐波（其中强调了第二个谐波）。

```
f = (0:n-1)*(fs/n)/10;
power = abs(y).^2/n;

plot(f(1:floor(n/2)),power(1:floor(n/2)))
xlabel('Frequency')
ylabel('Power')
```



另请参阅

[ifft](#) | [fft2](#) | [fftn](#) | [fft](#) | [fftshift](#) | [nextpow2](#)

相关示例

- “傅里叶变换” (第 16-2 页)
- “二维傅里叶变换” (第 16-17 页)

使用 FFT 进行多项式插值

使用快速傅里叶变换 (FFT) 来估算用于对一组数据进行插值的三角函数多项式的系数。

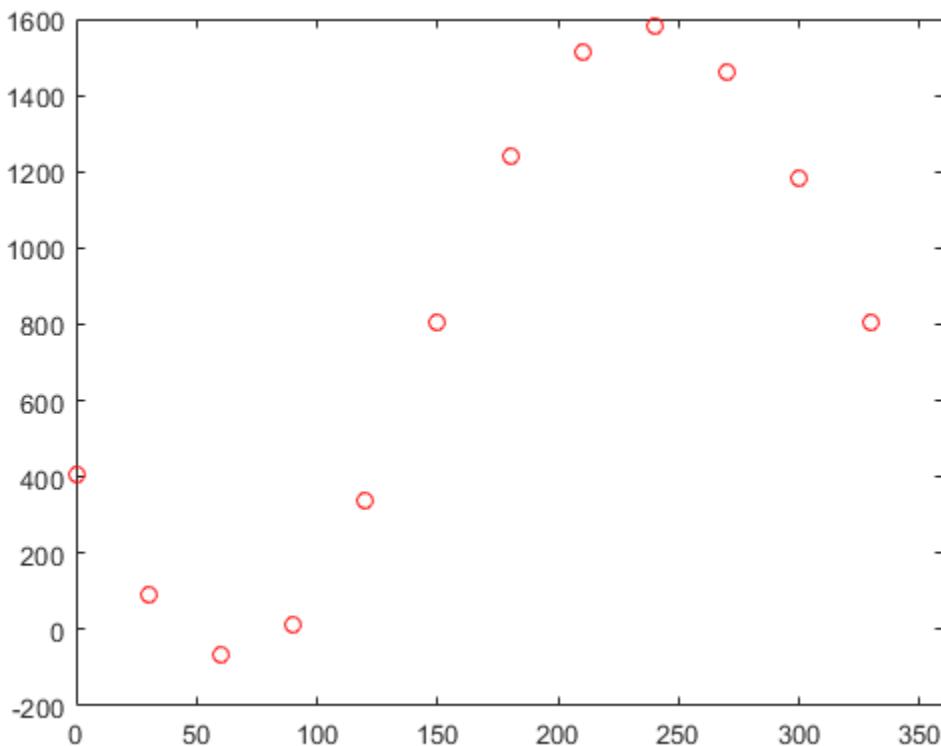
数学中的 FFT

FFT 算法通常与信号处理应用相关，但也可以在数学领域更广泛地用作快速计算工具。例如，通常通过直接解算线性方程组来计算用于对一组数据进行插值的 n 次多项式 $c_1x^n + c_2x^{n-1} + \dots + c_nx + c_{n+1}$ 的系数 c_i 。在 19 世纪初研究小行星轨道时，卡尔·弗里德里希·高斯发现了一种数学捷径，即通过将问题分解为多个较小的子问题，然后将结果合并来计算多项式插值的系数。他的方法等同于估算其数据的离散傅里叶变换。

小行星数据插值

在高斯的论文中，他描述了一种估算小行星智神星轨道的方法。他从以下 12 个二维数据点 x 和 y 开始。

```
x = 0:30:330;
y = [408 89 -66 10 338 807 1238 1511 1583 1462 1183 804];
plot(x,y,'ro')
xlim([0 360])
```



高斯使用以下形式的三角函数多项式为小行星的轨道建模。

$$\begin{aligned}
 y = & a_0 + a_1\cos(2\pi(x/360)) + b_1\sin(2\pi(x/360)) \\
 & a_2\cos(2\pi(2x/360)) + b_2\sin(2\pi(2x/360)) \\
 & \dots \\
 & a_5\cos(2\pi(5x/360)) + b_5\sin(2\pi(5x/360)) \\
 & a_6\cos(2\pi(6x/360))
 \end{aligned}$$

使用 **fft** 计算多项式的系数。

```

m = length(y);
n = floor((m+1)/2);
z = fft(y)/m;

a0 = z(1);
an = 2*real(z(2:n));
a6 = z(n+1);
bn = -2*imag(z(2:n));

```

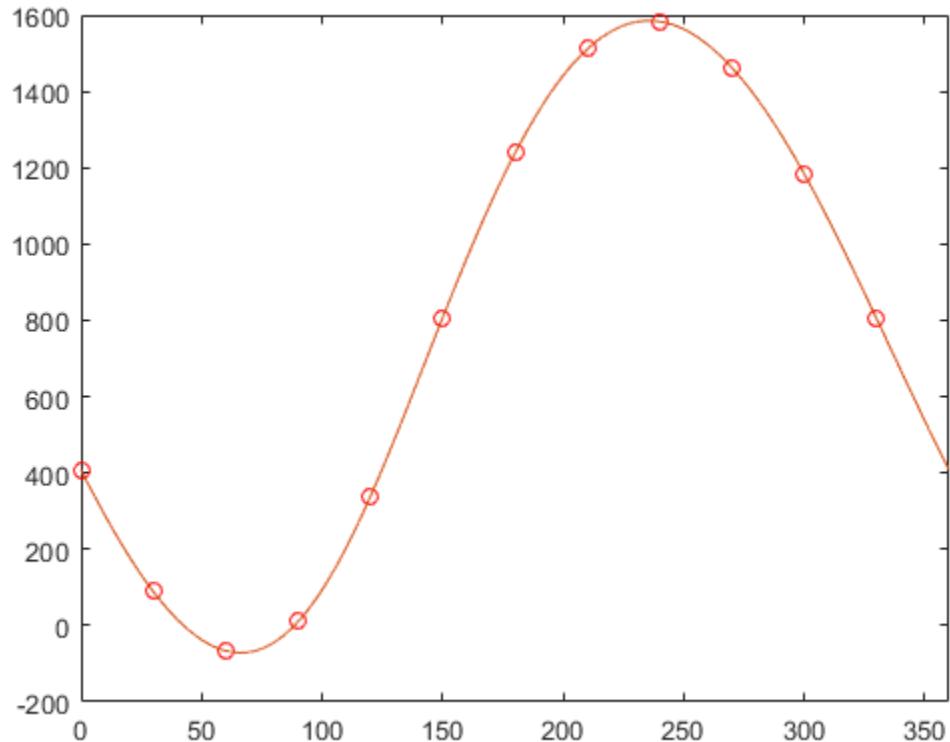
在原始数据点上绘制插值多项式。

```

hold on
px = 0:0.01:360;
k = 1:length(an);
py = a0 + an*cos(2*pi*k'*px/360) ...
    + bn*sin(2*pi*k'*px/360) ...
    + a6*cos(2*pi*6*px/360);

plot(px,py)

```



参考

- [1] Briggs, W. and V.E. Henson. *The DFT: An Owner's Manual for the Discrete Fourier Transform.* Philadelphia: SIAM, 1995.
- [2] Gauss, C. F. "Theoria interpolationis methodo nova tractata." *Carl Friedrich Gauss Werke.* Band 3. Göttingen: Königlichen Gesellschaft der Wissenschaften, 1866.
- [3] Heideman M., D. Johnson, and C. Burrus. "Gauss and the History of the Fast Fourier Transform." *Arch. Hist. Exact Sciences.* Vol. 34. 1985, pp. 265–277.
- [4] Goldstine, H. H. *A History of Numerical Analysis from the 16th through the 19th Century.* Berlin: Springer-Verlag, 1977.

另请参阅

fft

相关示例

- "傅里叶变换" (第 16-2 页)

二维傅里叶变换

`fft2` 函数将二维数据变换为频率空间。例如，您可以变换二维光学掩模以揭示其衍射模式。

二维傅里叶变换

以下公式定义 $m \times n$ 矩阵 X 的离散傅里叶变换 Y 。

$$Y_{p+1, q+1} = \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} \omega_m^{jp} \omega_n^{kq} X_{j+1, k+1}$$

ω_m 和 ω_n 是以下方程所定义的复单位根。

$$\omega_m = e^{-2\pi i/m}$$

$$\omega_n = e^{-2\pi i/n}$$

i 是虚数单位， p 和 j 是值范围从 0 到 $m-1$ 的索引， q 和 k 是值范围从 0 到 $n-1$ 的索引。在此公式中， X 和 Y 的索引平移 1 位，以反映 MATLAB 中的矩阵索引。

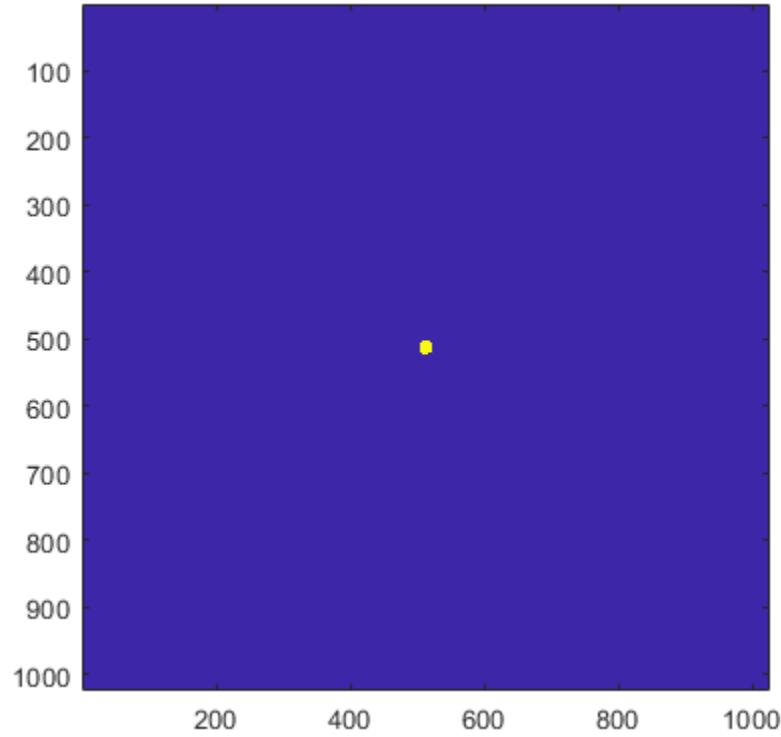
计算 X 的二维傅里叶变换等同于首先计算 X 每列的一维变换，然后获取每行结果的一维变换。换言之，命令 `fft2(X)` 等同于 `Y = fft(fft(X).').'`。

二维衍射模式

在光学领域，傅里叶变换可用于描述平面波入射到带有小孔的光学掩模上所产生的衍射模式 [1]。本示例对光学掩模使用 `fft2` 函数来计算其衍射模式。

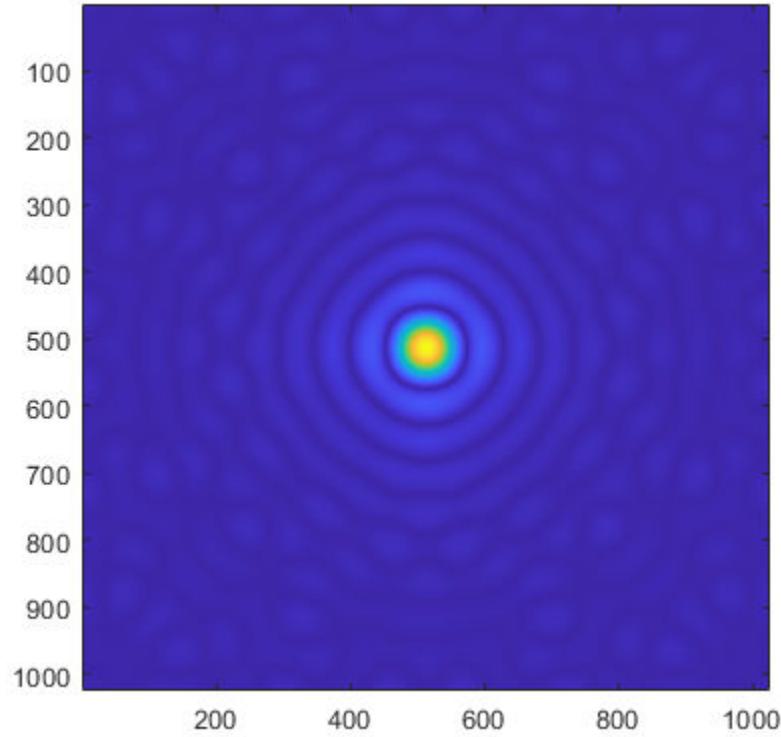
创建用于定义带有小圆孔的光学掩模的逻辑数组。

```
n = 2^10;           % size of mask
M = zeros(n);
I = 1:n;
x = I-n/2;         % mask x-coordinates
y = n/2-I;         % mask y-coordinates
[X,Y] = meshgrid(x,y); % create 2-D mask grid
R = 10;             % aperture radius
A = (X.^2 + Y.^2 <= R.^2); % circular aperture of radius R
M(A) = 1;           % set mask elements inside aperture to 1
imagesc(M)          % plot mask
axis image
```



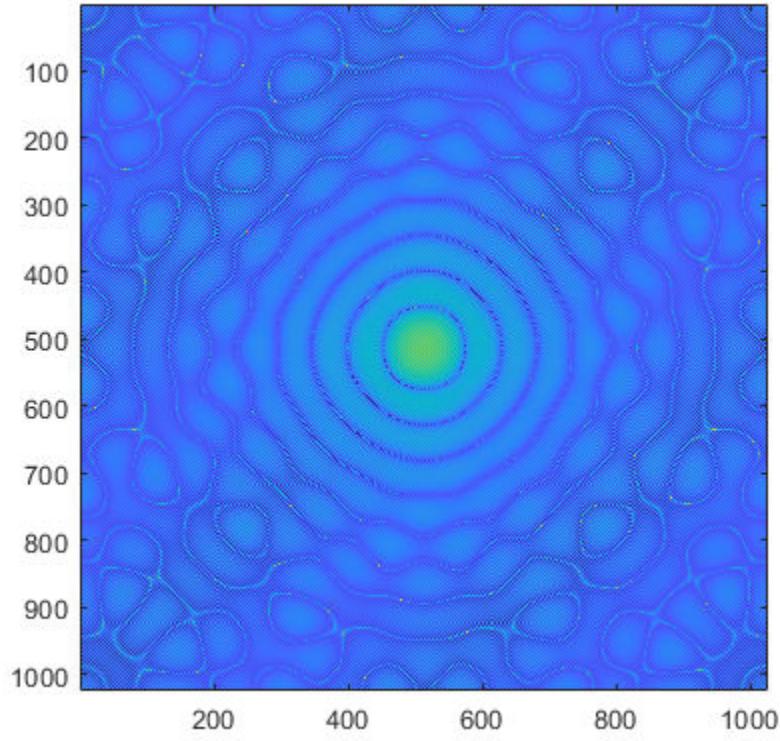
使用 `fft2` 计算掩模的二维傅里叶变换，并使用 `fftshift` 函数重新排列输出，从而使零频率分量位于中央。绘制生成的衍射模式频率。蓝色指示较小的幅值，黄色指示较大的幅值。

```
DP = fftshift(fft2(M));
imagesc(abs(DP))
axis image
```



为增强小幅值区域的细节，需绘制衍射模式的二维对数。极小的幅值会受数值舍入误差影响，而矩形网格则会导致径向非对称性。

```
imagesc(abs(log2(DP)))
axis image
```



参考

[1] Fowles, G. R. *Introduction to Modern Optics*. New York: Dover, 1989.

另请参阅

[fft](#) | [fft2](#) | [fftn](#) | [fftshift](#) | [ifft2](#)

相关示例

- “傅里叶变换” (第 16-2 页)

使用 FFT 进行频谱分析

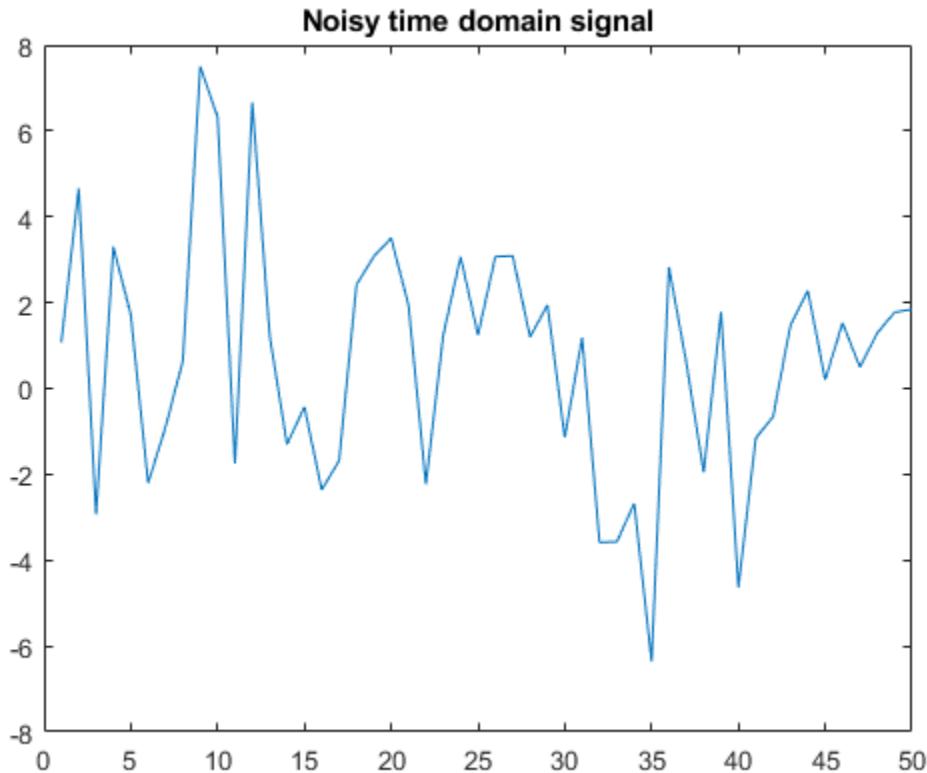
下面的示例说明了如何使用 FFT 函数进行频谱分析。FFT 的一个常用场景是确定一个时域含噪信号的频率分量。

首先创建一些数据。假设是以 1000 Hz 的频率对数据进行的采样。首先为数据构造一条时间轴，时间范围从 $t = 0$ 至 $t = 0.25$ ，步长为 1 毫秒。然后，创建一个包含 50 Hz 和 120 Hz 频率的正弦波信号 x 。

```
t = 0:0.001:0.25;
x = sin(2*pi*50*t) + sin(2*pi*120*t);
```

添加一些标准差为 2 的随机噪声以产生含噪信号 y 。然后，通过对该含噪信号 y 绘图来了解该信号。

```
y = x + 2*randn(size(t));
plot(y(1:50))
title('Noisy time domain signal')
```



很明显，通过观察该信号很难确定频率分量；这就是频谱分析为什么被广泛应用的原因。

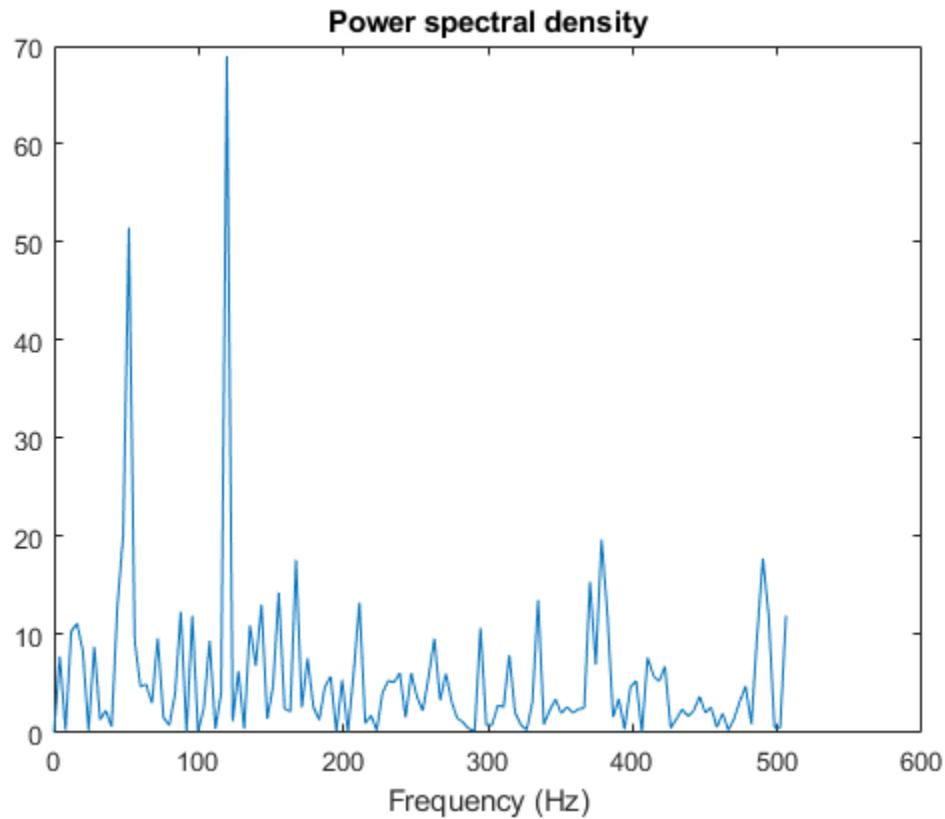
得到含噪信号 y 的离散傅里叶变换很容易；执行快速傅里叶变换 (FFT) 即可实现。

```
Y = fft(y,251);
```

使用复共轭 (CONJ) 计算功率谱密度，即测量不同频率下的能量。为前 127 个点构造一个频率轴，并使用该轴绘制结果图形。（其余的点是对称的。）

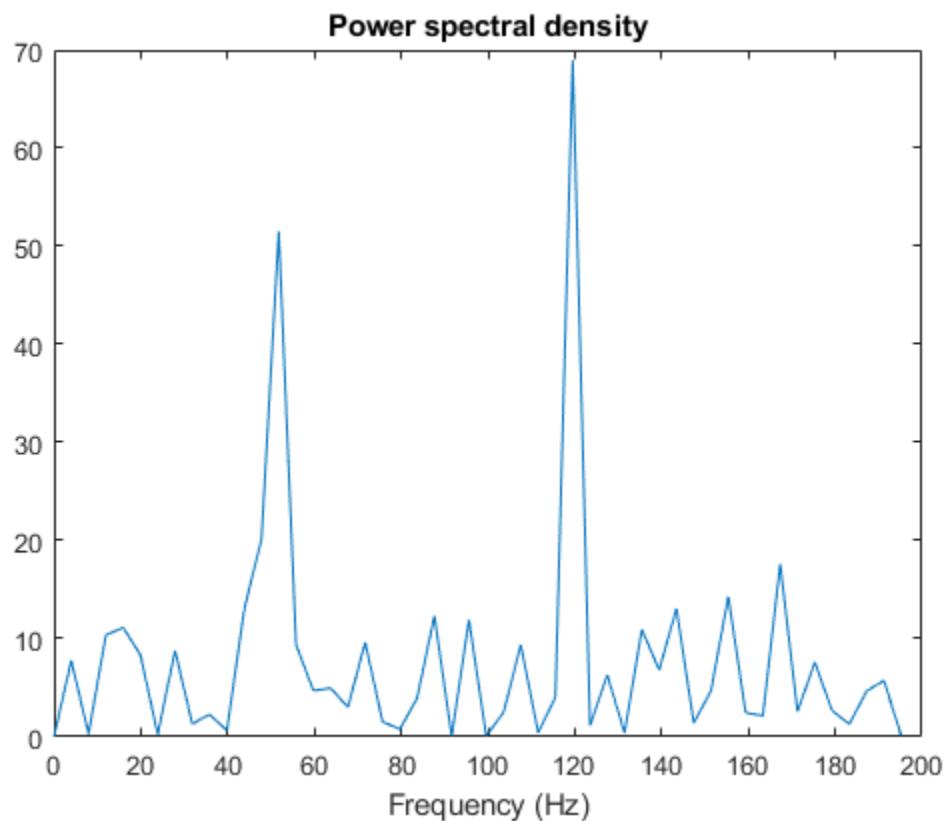
```
Pyy = Y.*conj(Y)/251;
f = 1000/251*(0:127);
```

```
plot(f,Pyy(1:128))
title('Power spectral density')
xlabel('Frequency (Hz)')
```



放大并仅绘制上限为 200 Hz 的图形。请注意 50 Hz 和 120 Hz 下的峰值。以下是原始信号的频率。

```
plot(f(1:50),Pyy(1:50))
title('Power spectral density')
xlabel('Frequency (Hz)')
```

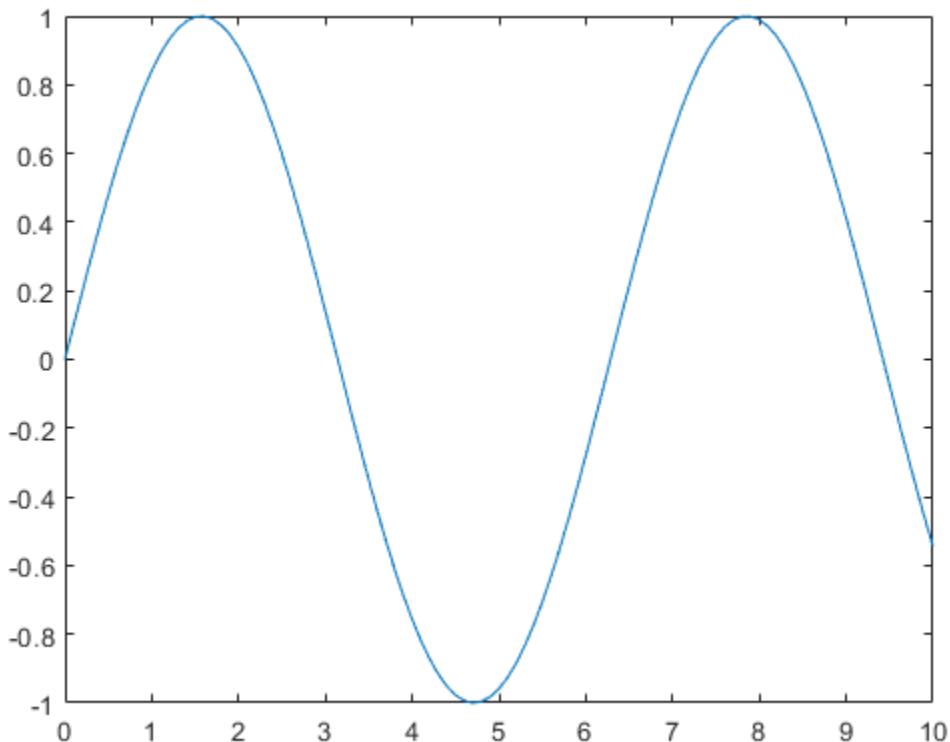


从正弦波转换为方波

此示例说明方波的傅里叶级数展开式是如何由奇次谐波的和构成的。

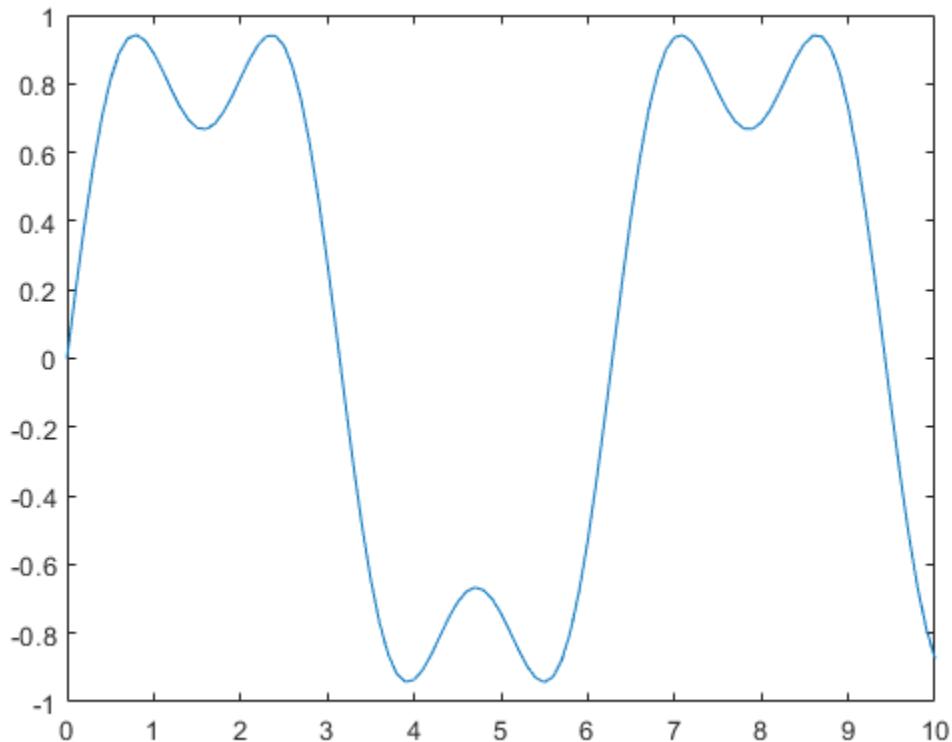
首先以 0.1 为步长，生成一个从 0 到 10 的时间向量，并求出所有点的正弦。绘制基频图。

```
t = 0:0.1:10;  
y = sin(t);  
plot(t,y);
```



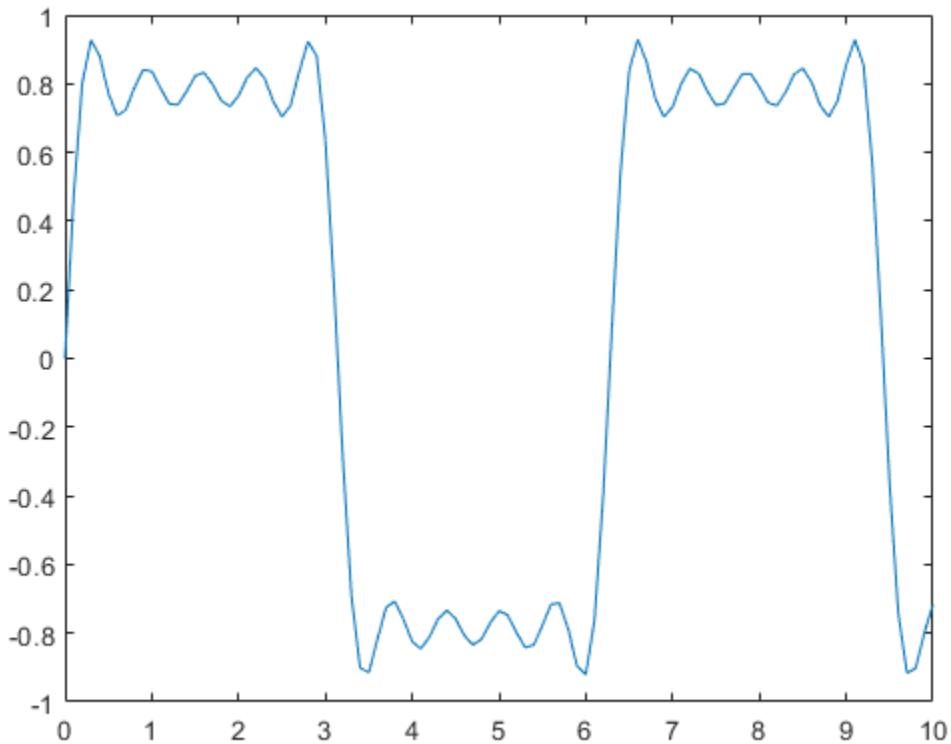
接下来，向基频添加第三个谐波，并绘制谐波图。

```
y = sin(t) + sin(3*t)/3;  
plot(t,y);
```



接下来使用第一、第三、第五、第七和第九个谐波。

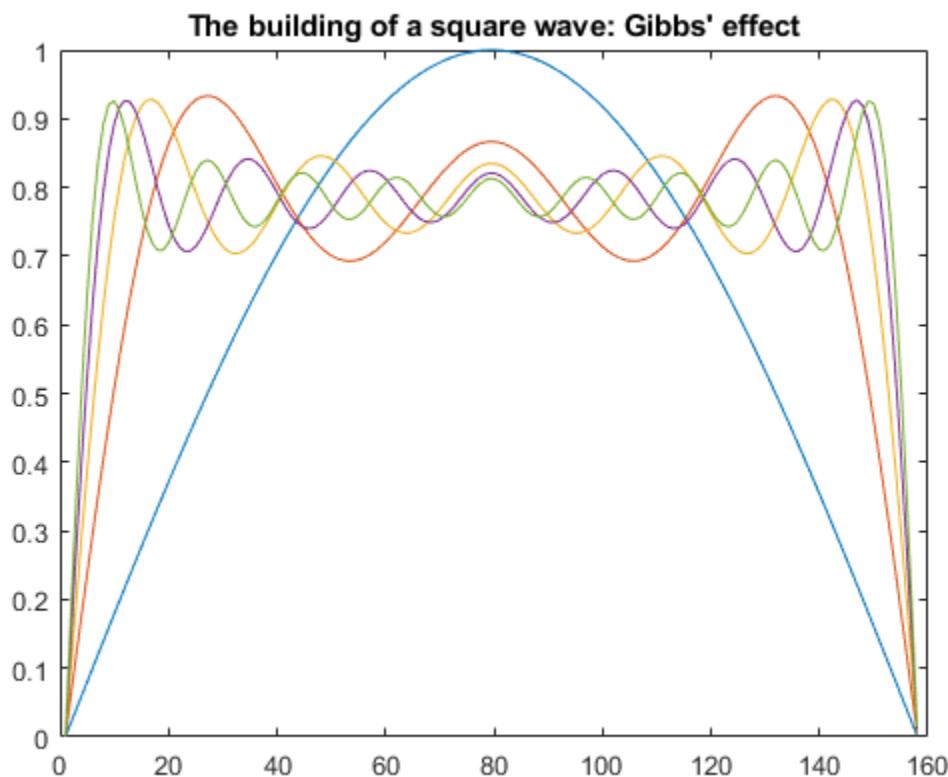
```
y = sin(t) + sin(3*t)/3 + sin(5*t)/5 + sin(7*t)/7 + sin(9*t)/9;  
plot(t,y);
```



最后，从基频开始创建更多连续谐波的向量，一直到第 19 个谐波为止，并将所有中间步长保存为矩阵的行。

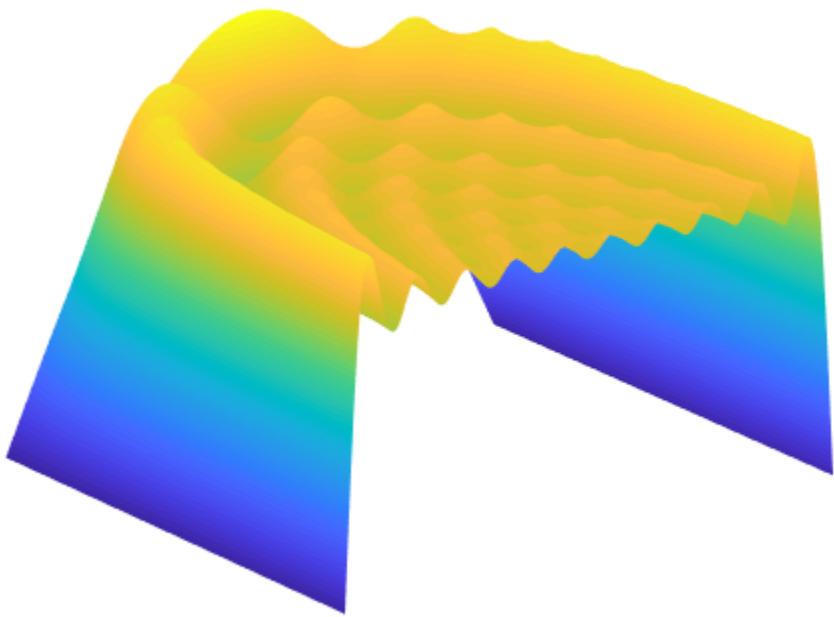
在同一个图窗中绘制这些向量，以便显示方波的演变。请注意，吉布斯效应表明它实际上永远不会转换为方波。

```
t = 0:0.02:3.14;
y = zeros(10,length(t));
x = zeros(size(t));
for k = 1:2:19
    x = x + sin(k*t)/k;
    y((k+1)/2,:) = x;
end
plot(y(1:2:9,:))
title('The building of a square wave: Gibbs'' effect')
```



下面提供了一个三维曲面图，该曲面图表示正弦波到方波的逐变过程。

```
surf(y);
shading interp
axis off ij
```

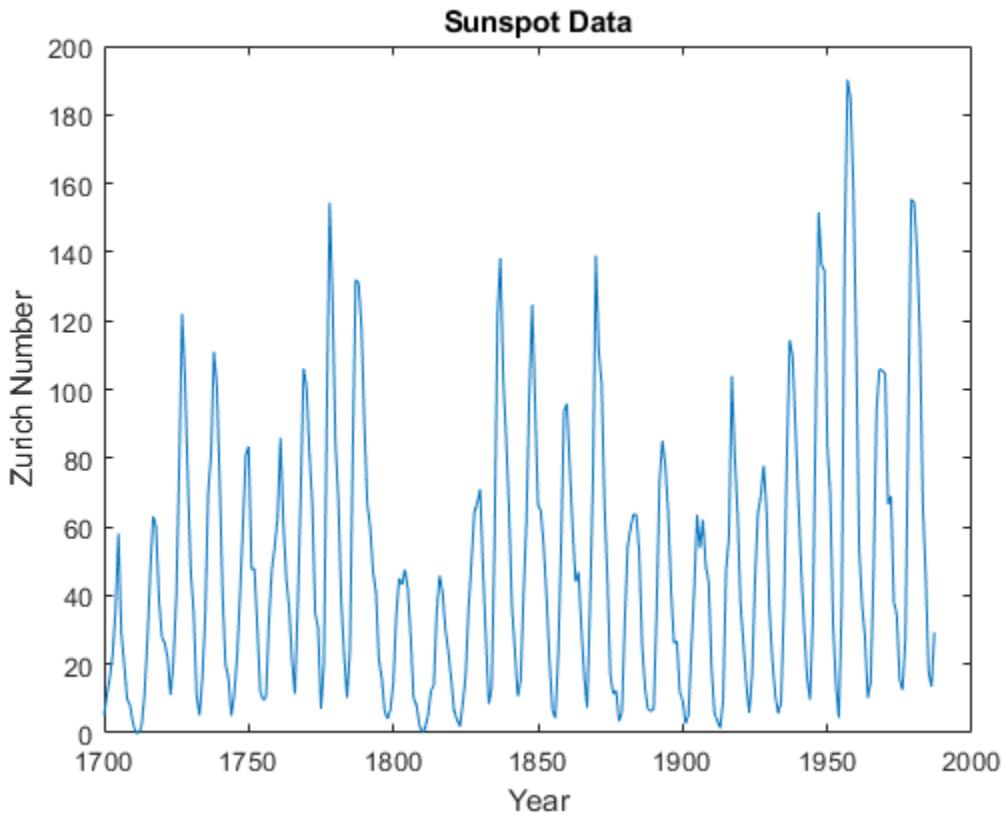


使用 FFT 分析周期性数据

可以使用傅里叶变换来分析数据中的变化，例如一个时间段内的自然事件。

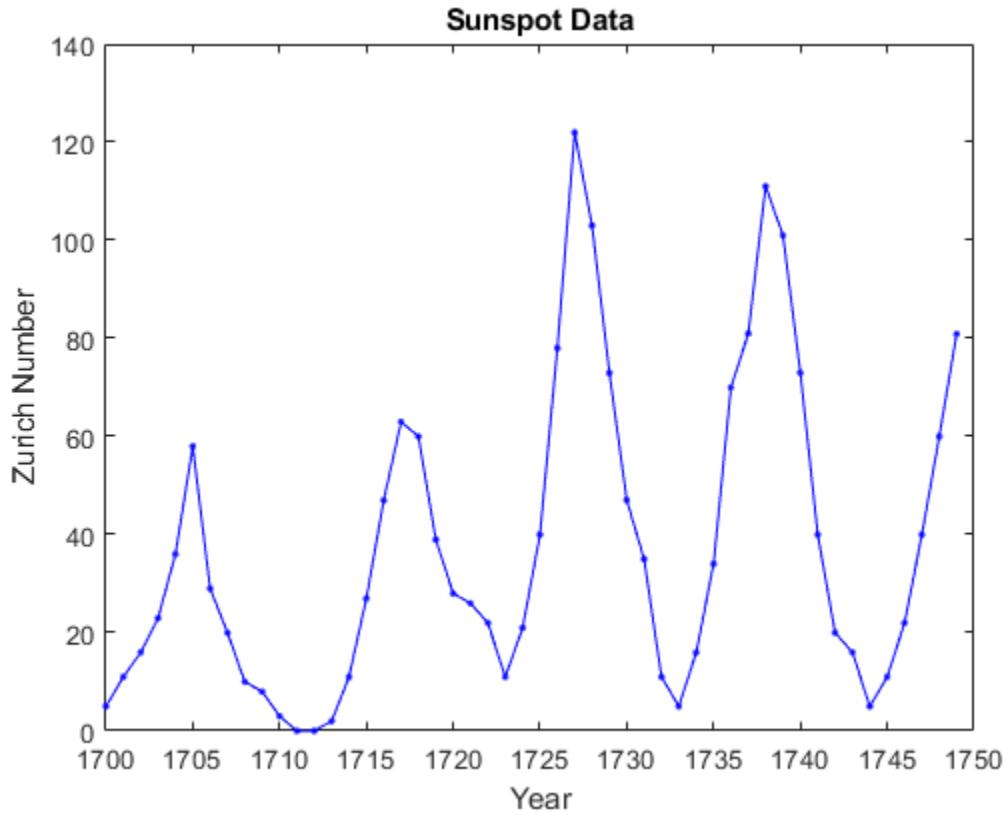
天文学家使用苏黎世太阳黑子相对数将几乎 300 年的太阳黑子的数量和大小制成表格。对大约 1700 至 2000 年间的苏黎世数绘图。

```
load sunspot.dat
year = sunspot(:,1);
relNums = sunspot(:,2);
plot(year,relNums)
xlabel('Year')
ylabel('Zurich Number')
title('Sunspot Data')
```



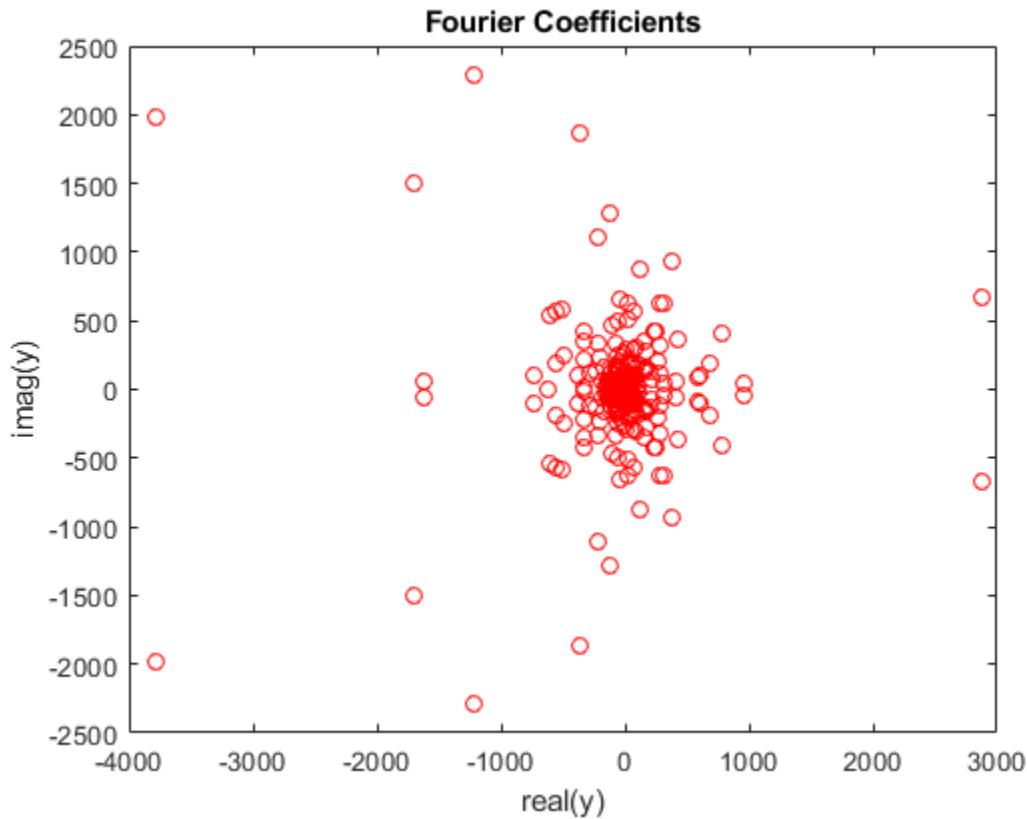
为了更详细地看太阳黑子活动的周期特性，将对前 50 年的数据绘图。

```
plot(year(1:50),relNums(1:50),'b.-');
xlabel('Year')
ylabel('Zurich Number')
title('Sunspot Data')
```



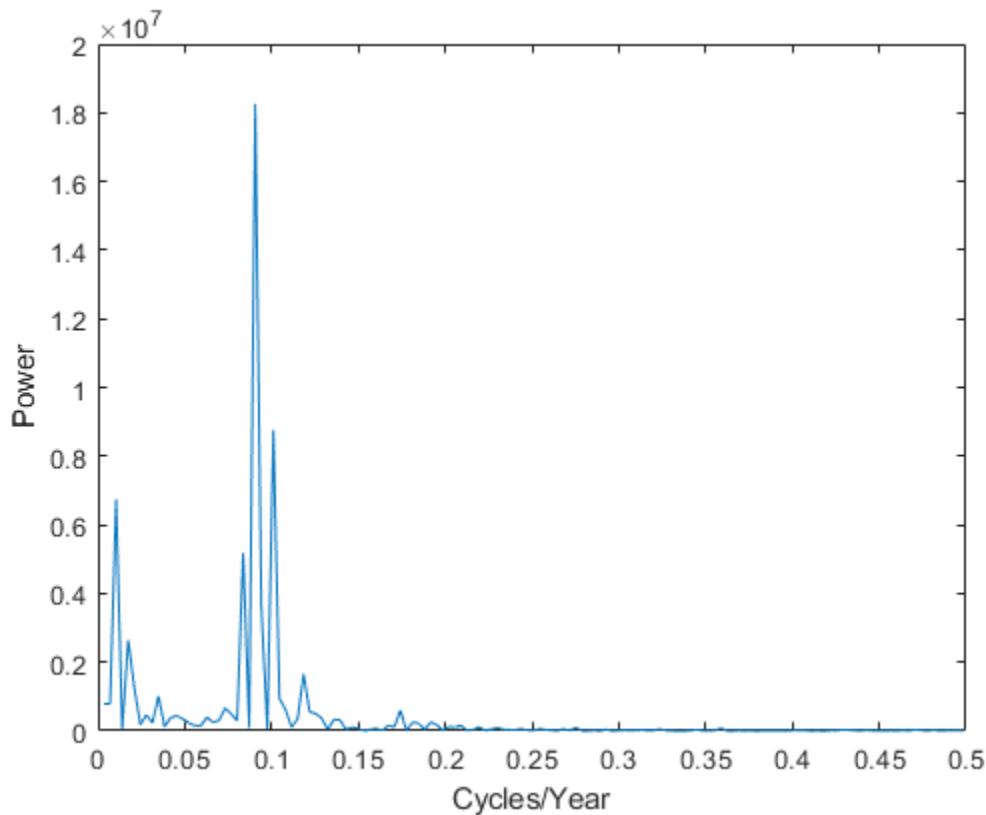
傅里叶变换是一种基础的信号处理工具，可确定数据中的频率分量。使用 `fft` 函数获取苏黎世数据的傅里叶变换。删除存储数据总和的输出的第一个元素。绘制该输出的其余部分，其中包含复傅里叶系数关于实轴的镜像图像。

```
y = fft(relNums);
y(1) = [];
plot(y,'ro')
xlabel('real(y)')
ylabel('imag(y)')
title('Fourier Coefficients')
```



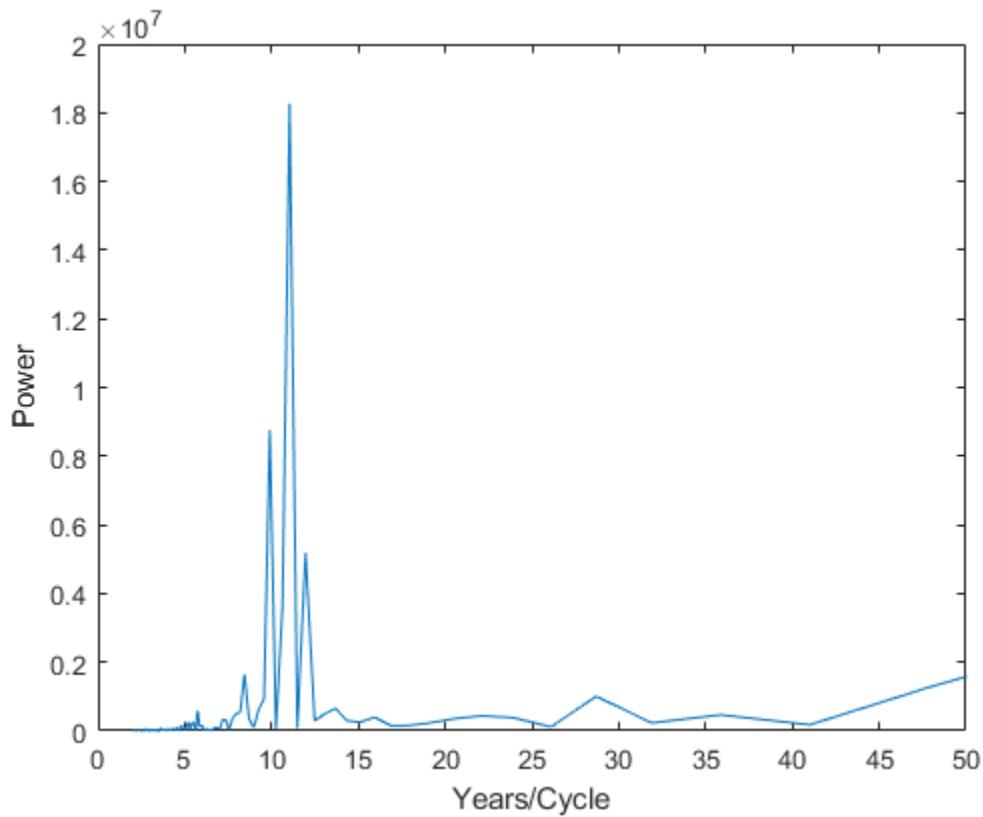
单独的傅里叶系数难以解释。计算系数更有意义的方法是计算其平方幅值，即计算幂。由于一半的系数在幅值中是重复的，因此您只需要对一半的系数计算幂。以频率函数的形式绘制功率谱图，以每年的周期数为测量单位。

```
n = length(y);
power = abs(y(1:floor(n/2))).^2; % power of first half of transform data
maxfreq = 1/2; % maximum frequency
freq = (1:n/2)/(n/2)*maxfreq; % equally spaced frequency grid
plot(freq,power)
xlabel('Cycles/Year')
ylabel('Power')
```



太阳黑子活动发生的大约每 11 年一次。为了查看更易解释的周期活动，以周期函数形式绘制幂图，以每周期的年数为测量单位。该绘图揭示了太阳黑子活动约每 11 年出现一次高峰。

```
period = 1./freq;
plot(period,power);
xlim([0 50]); %zoom in on max power
xlabel('Years/Cycle')
ylabel('Power')
```



另请参阅

[fft](#) | [fft2](#) | [fftw](#)

相关示例

- “傅里叶变换” (第 16-2 页)
- “二维傅里叶变换” (第 16-17 页)

