minoptlab 与科学计算

minoptlab.com

08/21/2025

Contents

1 前言

《minoptlab 与科学计算》是一本使用 minoptlab 介绍科学计算方法的入门书籍。

minoptlab 是一款通用科学计算软件，它提供丰富的数学计算函数库和强大的数学绘图功能，能够胜任复杂的数学计算和数据可视化任务。

本书将介绍以下内容。

- 介绍 minoptlab
- 线性方程组
- 插值
- 非线性方程组
- 微分和积分
- 最优化
- 最小二乘法
- 常微分方程组
- 随机数与统计
- 傅立叶变换
- 矩阵分解
- 绘图

本书的所有代码均能够在 minoptlab 上直接运行。

minoptlab 软件可从 https://minoptlab.com 下载。

2 介绍 minoptlab

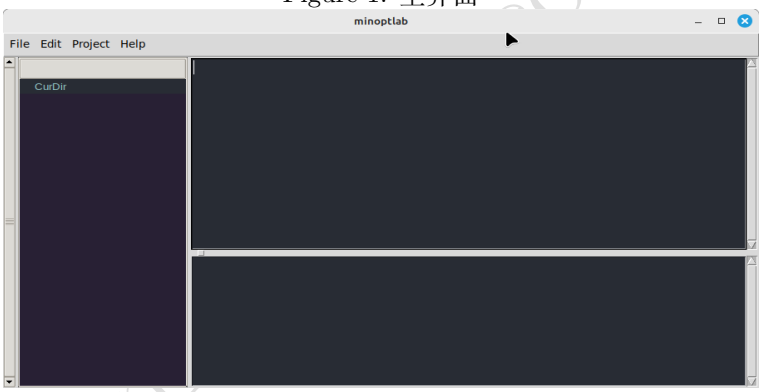
minoptlab 是长沙市先验真理数据科技有限公司推出的一款单机版通用科学计算软件，当前版本号为 1.0.0。LINUX 和 WINDOWS 系统均可安装使用。

该软件为商业软件，付费模式为月付 50 元人民币。其价格非常亲民，月成本仅仅相当于看一场电影。

尽管 minoptlab 价格不贵，但是其功能非常强大，易学易用，付费即享受全部功能，绝无功能限制。

minoptlab 语言的表层是 lua[1]，深层则是 c/c++/fortran，可谓易用易学与迅捷高效兼备。

Figure 1: 主界面



主界面左边是当前工作目录文件列表区。右边上半区是脚本输入区，下半区是脚本运行输出区。工作目录的选择，可通过点击 Project↦WorkDir 自由选择，需要确保路径名不能包含中文。

当前工作目录文件列表区会列出当前工作路径下的所有 lua 脚本文件（文件扩展名是*.lua）。点击列表中任一文件名，即可在右侧上半部分的脚本输入区加载该文件；点击 Porject↦Run 即可运行该脚本。

minoptlab 配备较详细的帮助文档和实例演示脚本。

点击 Help↦Doc 可打开帮助文档，点击 Help↦Sample_1 和 Help↦Sample_2 中的任意文件名，可加载对应实例演示脚本。

[1] https://lua.ac.cn/manual/5.4/

minoptlab 的数据拟合功能通过了 NIST（美国国家标准与技术研究院）公布的全部非线性拟合测试。点击 Help↦NIST 中的任一文件名，可加载对应测试脚本并自行完成运行测试。

2.1 数学函数

lua 内置 math 库，可提供一些常用的数学函数。如果需要更多，则可以调用 minoptlab 内置的 mathx 库。这个库包含以下数学函数：

```
mathx library:
 acos        cosh        fmax        lgamma      remainder
 acosh       deg         fmin        log         round
 asin        erf         fmod        log10       scalbn
 asinh       erfc        frexp       log1p       sin
 atan        exp         gamma       log2        sinh
 atan2       exp2        hypot       logb        sqrt
 atanh       expm1       isfinite    modf        tan
 cbrt        fabs        isinf       nearbyint   tanh
 ceil        fdim        isnan       nextafter   trunc
 copysign    floor       isnormal    pow         version
 cos         fma         ldexp       rad
```

其中有些函数是 math 库没有的。

2.2 复数

lua 自身不支持复数。minoptlab 内置了复数类：

```
complex library:
I       __tostring(z)   asinh(z)    imag(z)     sinh(z)
__add(z,w)  __unm(z)    atan(z)     log(z)      sqrt(z)
__div(z,w)  abs(z)      atanh(z)    new(x,y)    tan(z)
__eq(z,w)   acos(z)     conj(z)     pow(z,w)    tanh(z)
__mul(z,w)  acosh(z)    cos(z)      proj(z)     tostring(z)
__pow(z,w)  arg(z)      cosh(z)     real(z)     version
__sub(z,w)  asin(z)     exp(z)      sin(z)
```

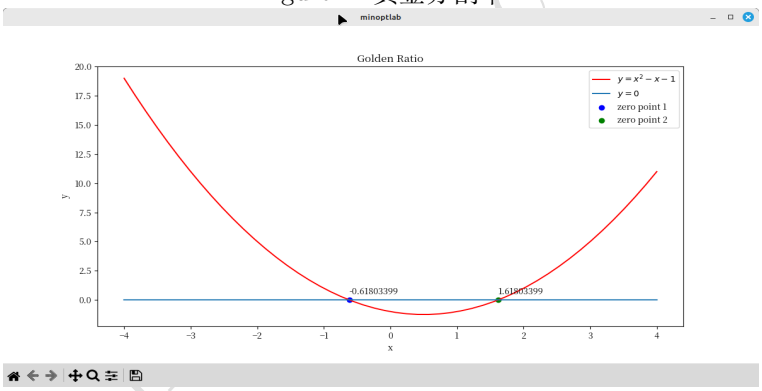
2.3 黄金分割

黄金分割率可视作一元二次方程 $x^2 - x - 1 = 0$ 的一个解。它有两个实解，这两个解在几何上就是曲线 $y = x^2 - x - 1$ 与直线 $y = 0$ 的两个交点的 x 坐标。

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local p = {-1, -1, 1}
local root = mol.roots(p)
print(root)

function zero(x)
    return 0
end

function poly(x)
    return mol.polyval(p, {x})[1]
end

local lb = {-4}
local ub = {4}

local img1 = plt.fplot({poly, zero}, {lb, lb}, {ub, ub}, {'$y=x^2-x-1$', '$y=0$'}, 50, {{
    color = 'r'
}})

local img2 = plt.scatter({{root[1], 0}, {root[2], 0}}, {'zero point 1', 'zero point 2'},
    {{
    color = 'b',
    marker = 'o'
}, {
    color = 'g',
    marker = 'o'
}}, {{
    text = {
        args = {root[1], 0 + 0.5, string.format('%.8f', root[1])},
        kwargs = {}
```

```
    }
}, {
    text = {
        args = {root[2], 0 + 0.5, string.format('%.8f', root[2])},
        kwargs = {}
    }
}})

local fw = plt.framework(111, 'Golden Ratio', {'x', 'y'})

plt.show({{fw, {img1, img2}}})
```

Figure 2: 黄金分割率



一元多项式用升幂的列表表示。多项式 $x^2 - x - 1$ 就是`{-1,-1,1}`。要计算它的值，用 `polyval` 命令，第一个参数是升幂列表，第二个参数是变量取值列表，返回值也是列表，其元素对应于第二个参数的变量取值。多项式的实根用 `roots` 命令获取，其唯一参数是升幂列表。

2.4 黄金分割数列

黄金分割数列即斐波那契数列。

```
local print = require('package.print').print
local mol = libminoptlab

function fibonacci(n)
    if n == 1 then
        return 1
    end
    if n == 2 then
        return 2
    end
    return fibonacci(n - 1) + fibonacci(n - 2)
end

local t = mol.linspace(1, 12, 12)

print(mol.apply(t, fibonacci))
```

`linspace` 命令用于生成从第一个参数到第二个参数之间的等分列表，列表元素数量由第三个参数决定。`apply` 命令则能把作为第二个参数的一元函数，逐个作用于作为第一个参数的列表元素，并返回相应的结果列表。

3 线性方程组

线性方程组的一般表示是矩阵方程：$Ax = b$。它的解是 $x = A^{-1}b$。其中，A^{-1} 指的是矩阵 A 的逆矩阵。

在 minoptlab 中，矩阵的创建用 matirx 类，它的逆矩阵可调用 inv 方法求得。

对于 A 非方阵或者奇异的情况，可调用 pinv 方法获得 A 的伪逆矩阵 A^{+}。

因此，对于一般的线性方程组，$x = A^{+}b$ 总是可行的求解方法。这种方法对应于用 solve 方法来解。

```
1 local print = require('package.print').print
2 local mol = libminoptlab
3
4 local A = mol.matrix.new(3,3,{10,-3,5,-7,2,-1,0,6,5})
5 local b = mol.matrix.new(3,1,{7,4,6})
6 local x = A:solve(b)
7 print(x)
8 print(A:mul(x):sub(b):norm())
```

matrix 类数据的录入，必须按列优先的方式进行。比如矩阵：

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix}$$

录入时顺序是{10,-3,5,-7,2,-1,0,6,5}，而不是{10,-7,0,-3,2,6,5,-1,5}。

matrix 类的标量加减乘除分别对应于方法：add，sub，prod，div。矩阵乘法则对应于 mul 方法。inv 或 pinv 相当于矩阵除法。norm 方法给出矩阵的 2-范数。

对于大型数据，matrix 提供 read 和 write 方法读写 csv 数据文件。

4 插值

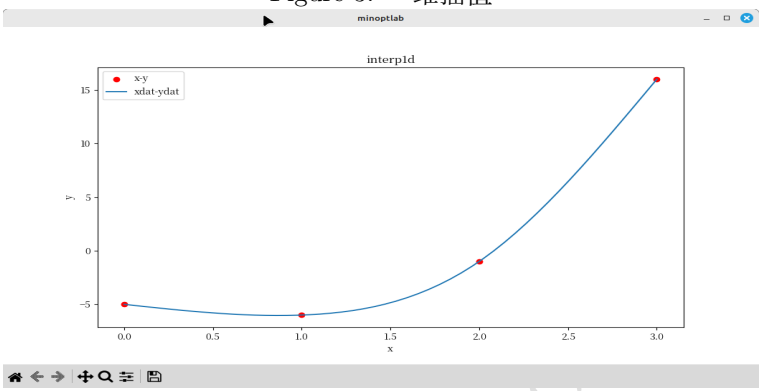
4.1 一维插值

一维插值用 interp1d 类。

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local x = mol.linspace(0, 3, 4)
local y = {-5, -6, -1, 16}

local f = mol.interp1d.new()
f:set(x, y)
local g = function(x)
    return f:get(x)
end

local xdat = mol.linspace(0, 3, 50)
local ydat = mol.apply(xdat, g)

local img1 = plt.scatter({{x, y}}, {'x-y'}, {{
    color = 'r'
}})
local img2 = plt.line({{xdat, ydat}}, {'xdat-ydat'})
local fw = plt.framework(111, 'interp1d', {'x', 'y'})
plt.show({{fw, {img1, img2}}})
```

interp1d 类只有两个方法：set 和 get。set 方法的两个参数分别是待插值的自变量数据列表和因变量数据列表。get 方法计算与给定的自变量参数数据相对应的因变量数据。

Figure 3: 一维插值



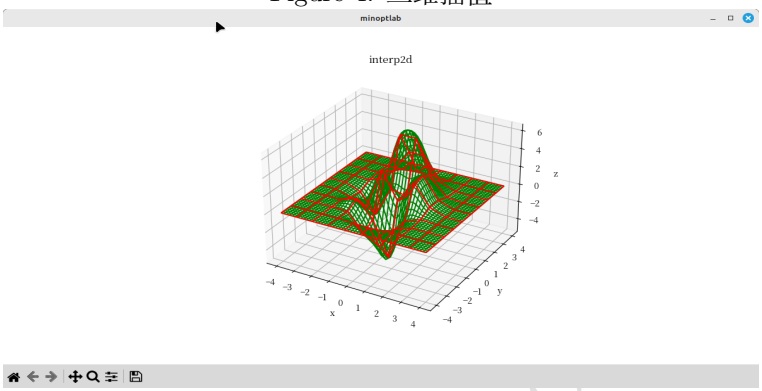
4.2 二维插值

二维插值用 interp2d 类。

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local peaks = function(xx)
    local x, y = xx[1], xx[2]
    return
        3 * (1 - x) ^ 2 * math.exp(-(x ^ 2) - (y + 1) ^ 2) - 10 * (x / 5 - x ^ 3 - y ^ 5)
     * math.exp(-x ^ 2 - y ^ 2) - 1 /
           3 * math.exp(-(x + 1) ^ 2 - y ^ 2)
end

local x = mol.linspace(-4, 4, 10)
local y = mol.linspace(-4, 4, 10)

local f = mol.interp2d.new()
f:set(x, y, peaks)
local g = function(x)
    return f:get(x)
end

local img1 = plt.line3d({peaks}, {{-4, -4}}, {{4, 4}}, 10, {{
    color = 'r'
}})
local img2 = plt.line3d({g}, {{-4, -4}}, {{4, 4}}, 50, {{
    color = 'g'
}})
local fw = plt.framework3d(111, 'interp2d', {'x', 'y', 'z'})
plt.show({{fw, {img2, img1}}})
```

interp2d 类也只有两个方法：set 和 get。set 方法的三个参数分别是待插值的第一、二维自变量数据列表和二元采样函数。get 方法计算与给定的自变量参数数据相对应的因变量数据。

Figure 4: 二维插值



5 非线性方程组

一元多项式方程求实根用 `roots`。比如一个用升幂列表表示的一元多项式`{-1, -1, 1}`，即 $x^2 - x - 1 = 0$。那么它的全部实根可以用命令 `roots({-1,-1,1})` 直接获得。

```
local print = require('package.print').print
local mol = libminoptlab

print(mol.roots({-1, -1, 1}));
```

对于一般的非线性方程组求根，可以使用 `findroot`，`fsolve` 或者 `xfsolve`。

看例子。

$$\exp\left(-\exp\left(-\left(x_1+x_2\right)\right)\right)-x_2\left(1+x_1^2\right)=0$$
$$x_1\cos\left(x_2\right)+x_2\sin\left(x_1\right)-0.5=0$$

```
local print = require('package.print').print
local mol = libminoptlab

function eq(x)
    return {math.exp(-math.exp(-(x[1] + x[2]))) - x[2] * (1 + x[1] ^ 2),
            x[1] * math.cos(x[2]) + x[2] * math.sin(x[1]) - 0.5}
end
local lb = {-10, -10}
local ub = {10, 10}

local root = mol.findroot(eq, lb, ub)

print({
    root = root,
    eq = eq(root)
})

local x0 = {1, 1}

root = mol.fsolve(eq, lb, ub, x0)
print({
    root = root,
    eq = eq(root)
```

```
})


local eps = 1e-5
local n = 200
root = mol.xfsolve(eq, x0, eps, n)

print({
    root = root,
    eq = eq(root)
})
```

findroot 命令的三个参数分别是方程和变量的搜索下界和上界。搜索界限用列表表示，方程用函数表示。

fsolve 命令比 findroot 多一个猜想值参数，该参数用列表表示。

xfsolve 命令不需要搜索界限参数，但需要猜想值、收敛精度和最大迭代次数参数。它使用牛顿-雅可比迭代法求根，底层用 c++实现。其实也可以在 minoptlab 上用 lua 实现：

```
local print = require('package.print').print
local mol = libminoptlab

function newton(eq, x0, eps, n)
    local ret = {table.unpack(x0)}
    local dim = #ret
    local t = 0
    for i = 1, n do
        local jv = mol.jaco(eq, ret)
        local fv = mol.matrix.new(eq(ret))
        local dx = jv:solve(fv:prod(-1))
        for j = 1, dim do
            ret[j] = ret[j] + dx:get(j)
        end
        if fv:norm() < eps then
            t = i
            break
        end
    end
    return ret, t
end
```

```
22
23 function eq(x)
24     return {math.exp(-math.exp(-(x[1] + x[2]))) - x[2] * (1 + x[1] ^ 2),
25             x[1] * math.cos(x[2]) + x[2] * math.sin(x[1]) - 0.5}
26 end
27
28 local x0 = {1, 1}
29 local eps = 1e-5
30 local n = 200
31 local root, t = newton(eq, x0, eps, n)
32
33 print({
34     root = root,
35     eq = eq(root),
36     t = t
37 })
```

函数 newton 返回根和迭代次数。该函数的实现用到了计算雅可比矩阵的 jaco 命令和求解线性方程组的 solve 命令。

minoptlab 提供方便计算函数的一阶、二阶信息的三个命令：grad，jaco 和 hess，分别对应于计算梯度、雅可比矩阵和海塞矩阵。在数值算法编程中，它们是极有用的工具。

6 微分与积分

6.1 一阶与二阶信息

梯度、雅可比矩阵和海塞矩阵分别用 grad，jaco 和 hess 获取。

```
local print = require('package.print').print
local mol = libminoptlab

function f1(x)
    return x[2] * math.sin(x[1]) + x[1] * math.cos(x[2])
end

function f2(x)
    return {math.exp(-math.exp(-(x[1] + x[2]))) - x[2] * (1 + x[1] ^ 2),
            x[1] * math.cos(x[2]) + x[2] * math.sin(x[1]) - 0.5}
end

local x = {0.5, 0.8}

print(mol.grad(f1, x))
print(mol.hess(f1, x))
print(mol.jaco(f2, x))
```

要注意的是：grad 和 hess 命令的第一个参数的函数形式与 jaco 命令是不同的；前者返回数值，表示的是一个数学函数，后者返回的是数值列表，表示的是一组数学函数。

6.2 一维积分与多维积分

一般的，一维和多维积分都可以用 integrate 命令。

```
local print = require('package.print').print
local mol = libminoptlab

function f1(x)
    return math.cos(x ^ 2 - 1) / x
end

function f2(x)
    return math.exp(-x)
end

function f3(x)
    return math.exp(x)
end

function f4(x)
    return math.exp(-x ^ 2)
end

function f5(x)
    local dim = #x
    local sum = 0
    for i = 1, dim do
        sum = sum + (x[i] - 0.5) ^ 2
    end
    return math.exp(-sum * 100) * 1013.2118364296088
end

print({
    f1 = mol.integrate(f1, -1, 2),
    f2 = mol.integrate(f2, 0, ''),
    f3 = mol.integrate(f3, '', 1),
    f4 = mol.integrate(f4, '', ''),
    f5 = mol.integrate(f5, {-1, 0, 0, 0}, {1, 1, 1, 1})
})
```

无穷大无论正负，一律用空字符串表示。对于多维的情况，积分上下限都用列表表示。

如果需要精度更高的积分，三维以内可以使用内置的 glint 包。这时，对于二维和三维的情况，积分区域的表达需要用函数，而非数值。

```
local print = require('package.print').print
local mol = libminoptlab

local glint = require('package.glint')

function f1(x)
    return math.log(x)
end

local lb1 = 0
local ub1 = 1
print({
    int1 = glint.int1d(f1, lb1, ub1)
})

function f2(x)
    return x[2] ^ 2 * math.sin(x[1] + x[2]) ^ 2 * math.cos(x[1])
end

local lb2 = {-math.pi / 2, function(x)
    return -math.pi
end}
local ub2 = {math.pi / 2, function(x)
    return math.pi
end}
print({
    int2 = glint.int2d(f2, lb2, ub2)
})

function f3(x)
    return math.sqrt(x[1] * x[2] * x[3])
end

local lb3 = {0, function(x)
    return 0
```

```
end, function(x)
    return math.sqrt(4 - x[1] ^ 2 - x[2] ^ 2)
end}
local ub3 = {2, function(x)
    return math.sqrt(4 - x ^ 2)
end, function(x)
    return 8 - x[1] - x[2]
end}
print({
    int3 = glint.int3d(f3, lb3, ub3)
})
```

7 最优化

7.1 线性规划

线性规划用 lpsolve 命令。

```
local print = require('package.print').print
local mol = libminoptlab

local c = {-6, 4, 1}
local Aeq = mol.matrix.new(1, 3, {0, -3, 2})
local beq = {6}
local A = mol.matrix.new(2, 3, {5, -1, -10, 3, 0, 0})
local b = {20, 3}
local lb = {-100, -100, -100}
local ub = {100, 100, 100}

local best, fbest = mol.lpsolve(c, Aeq, beq, A, b, lb, ub)

print({
    best = best,
    fbest = fbest,
    eqcheck = Aeq:mul(best):sub(beq):tovector(),
    ineqcheck = A:mul(best):sub(b):tovector()
})
```

其中，向量用列表表示，约束矩阵用 matrix 类表示。

如果是使用 mps 文件描述线性规划的情况，应该使用 lpmps 命令。比如下例：

```
NAME
ROWS
 N  Obj
 L  r0
 L  r1
 L  r2
 L  r3
 L  r4
 L  r5
 E  r6
COLUMNS
```

```
    c0        Obj       -1
    c0        r0        1
    c0        r1        1
    c0        r2        1
    c0        r3        -0.25
    c0        r4        -1
    c0        r5        -1
    c0        r6        1
    c1        Obj       -0.3333333333
    c1        r0        1
    c1        r1        0.25
    c1        r2        -1
    c1        r3        -1
    c1        r4        -1
    c1        r5        1
    c1        r6        0.25
RHS
    RHS_V     r0        2
    RHS_V     r1        1
    RHS_V     r2        2
    RHS_V     r3        1
    RHS_V     r4        -1
    RHS_V     r5        2
    RHS_V     r6        0.5
BOUNDS
 LO BOUND     c0        -1
 UP BOUND     c0        1.5
 LO BOUND     c1        -0.5
 UP BOUND     c1        1.25
ENDATA
```

```
local print = require('package.print').print
local mol = libminoptlab

local mps = 'lp.mps'

local best, fbest = mol.lpmps(mps)

print({
```

```
9     best = best,
10    fbest = fbest
11 })
```

lpmps 命令会读入文件，然后求解它。因此，它的唯一参数务必提供正确的文件地址。

7.1.1 供需平衡问题

供需平衡问题特指由以下模型界定的问题：

$$\begin{aligned} \textbf{min} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\ & \forall i, \sum_{i=1}^{m} x_{ij} = S_i \\ & \forall j, \sum_{j=1}^{n} x_{ij} = D_j \\ & \sum_{i=1}^{m} S_i \equiv \sum_{j=1}^{n} D_j \end{aligned}$$

minoptlab 为此类问题特别提供 `lpap` 命令。

比如，某公司从三个产地 A_i 将产品运往四个销地 B_j，各产地的产量、各销地的销量，以及各产地往各销地的运费单价如下表所示：

	B_1	B_2	B_3	B_4	供应
A_1	3	11	3	10	7
A_2	1	9	2	8	4
A_3	7	4	10	5	9
需求	3	6	5	6	20

要如何调运才能使得总运费最小?

```
local print = require('package.print').print
local mol = libminoptlab

local cost = mol.matrix.new(3,4,{3,1,7,11,9,4,3,2,10,10,8,5})
local eq = {7,4,9,3,6,5,6}

local lb = {}
local ub = {}
local n = cost.nrow*cost.ncol
for i = 1,n do
    table.insert(lb,0)
    table.insert(ub,100)
end

```

```
local best,fbest = mol.lpap(cost,eq,lb,ub)

print({
    best = best,fbest = fbest
})
print(mol.matrix(cost.nrow,cost.ncol,best))
```

费用矩阵用 matrix 类表示，等式条件用列表表示，顺序是先供给后需求。

供需平衡问题的一个特殊情况就是指派问题。所求皆为 0-1 变量。比如指派矩阵如下：

$$\begin{bmatrix} 6 & 7 & 11 & 2 \\ 4 & 5 & 9 & 8 \\ 3 & 1 & 10 & 4 \\ 5 & 9 & 8 & 2 \end{bmatrix}$$

```
local print = require('package.print').print
local mol = libminoptlab

local cost = mol.matrix.new(4, 4, {6, 4, 3, 5, 7, 5, 1, 9, 11, 9, 10, 8, 2, 8, 4, 2})
local eq = {}

local m = cost.nrow + cost.ncol
for i = 1, m do
    table.insert(eq, 1)
end

local n = cost.nrow * cost.ncol
local lb = {}
local ub = {}
for i = 1, n do
    table.insert(lb, 0)
    table.insert(ub, 1)
end

local best, fbest = mol.lpap(cost, eq, lb, ub)

print({
    best = best,
```

```
    fbest = fbest
})
print(mol.matrix(cost.nrow, cost.ncol, best))
```

对于供需不平衡的情况，可以通过添加虚拟需求或者虚拟供给的方法，来实现平衡。然后再求解。比如某公司有 6 个供货栈 (仓库), 库存货物总数分别为 60,55,51,43, 41,52, 现有 8 个客户各要一批货, 数量分别为 35,37,22,32,41,32,43, 38. 各供货栈到 8 个客户处的单位货物运输价见下表。试确定各货栈到各客户处的货物调运数量, 使总的运输费用最小。

	d1	d2	d3	d4	d5	d6	d7	d8	d9	S
s1	6	2	6	7	4	2	5	9	0	60
s2	4	9	5	3	8	5	8	2	0	55
s3	5	2	1	9	7	4	3	3	0	51
s4	7	6	7	3	9	2	7	1	0	43
s5	2	3	9	5	7	2	6	5	0	41
s6	5	5	2	2	8	1	4	3	0	52
D	35	37	22	32	41	32	43	38	22	302

其中，总需求 D 比总供给 S 少 22，故而新增了虚拟客户 d9，以实现供求平衡。

```
local print = require('package.print').print
local mol = libminoptlab

local cost = mol.matrix.new(6, 9,
    {6, 4, 5, 7, 2, 5, 2, 9, 2, 6, 3, 5, 6, 5, 1, 7, 9, 2, 7, 3, 9, 3, 5, 2, 4, 8, 7, 9,
    7, 8, 2, 5, 4, 2, 2, 1, 5, 8,
     3, 7, 6, 4, 9, 2, 3, 1, 5, 3, 0, 0, 0, 0, 0, 0})

local eq = {60, 55, 51, 43, 41, 52, 35, 37, 22, 32, 41, 32, 43, 38, 22}

local lb = {}
local ub = {}
local n = cost.nrow * cost.ncol
for i = 1, n do
    table.insert(lb, 0)
    table.insert(ub, 100)
end
```

```
17
18 local best, fbest = mol.lpap(cost, eq, lb, ub)
19
20 print({
21     best = best,
22     fbest = fbest
23 })
24 print(mol.matrix(cost.nrow, cost.ncol, best))
```

7.2 非线性规划

非线性规划的基本形式如下：

$$\begin{aligned} \textbf{min} \quad & f(x) \\ s.t. \quad & h(x) = 0 \\ & g(x) \leq 0 \\ & x \subseteq \boldsymbol{R}^n \\ & \forall i, i \in \{1, \cdots, n\}, x_i \in [L_i, U_i] \end{aligned}$$

minoptlab 规定优化方向为最小化，不等式约束为不大于约束。对于最大化问题，目标函数要乘以 -1，不小于约束要两边乘以 -1。

7.2.1 无约束

无约束的非线性规划一般用 `fmin` 求解。比如：

$$\begin{aligned} f(x) &= -\left(\frac{\sin(r)}{r} + 1\right) \\ r &= \sqrt{(x_1 - 50)^2 + (x_2 - 50)^2} + \exp(1) \end{aligned}$$

```
local print = require('package.print').print
local mol = libminoptlab

function f(x)
    local r = math.sqrt((x[1] - 50) ^ 2 + (x[2] - 50) ^ 2) + math.exp(1)
    return -(math.sin(r) / r + 1)
end

local lb = {0, 0}
local ub = {100, 100}
local eps = 1e-5
local n = 2000
local m = 20

local best, fbest = mol.fmin(f, lb, ub, eps, n, m)

print({
```

```
18     best = best,
19     fbest = fbest
20 })
```

fmin 命令的六个参数分别是目标函数、搜索下界、搜索上界，收敛精度、最大迭代次数和种群数。它使用进化算法求解，能最大限度保证获得全局最小解。

7.2.2 有约束

对于有约束条件的非线性规划，使用 fmincon 命令。比如：

$$
\begin{aligned}
f(x) &= \frac{\sin(2\pi x_1)^3 \sin(2\pi x_2)}{x_1^3(x_1+x_2)} \\
g_1(x) &= x_1^2 - x_2 + 1 \le 0 \\
g_2(x) &= 1 - x_1 + (x_2 - 4)^2 \le 0
\end{aligned}
$$

```
local print = require('package.print').print
local mol = libminoptlab

function f(x)
    local a = math.sin(2 * math.pi * x[1]) ^ 3 * math.sin(2 * math.pi * x[2])
    local b = x[1] ^ 3 * (x[1] + x[2])
    return -a / b
end

local lb = {0, 0}
local ub = {10, 10}

local h = mol.nullconfun()
local g = function(x)
    return {x[1] ^ 2 - x[2] + 1, 1 - x[1] + (x[2] - 4) ^ 2}
end

local eps = 1e-5
local algo = mol.listoptalgo()[1]
local n = 2000

local best, fbest = mol.fmincon(f, lb, ub, h, g, algo, eps, n)

print({
    best = best,
    fbest = fbest,
    ineqcon = g(best)
})
```

fmincon 命令的八个参数分别是目标函数、搜索下界、搜索上界、等式约束、不等式约束、算法、收敛精度和最大迭代次数。约束不存在时，用 nullconfun 命令创建空约束。算法选择有多种，

`listoptalgo` 命令会返回一个算法字符串列表，其中既有保证全局优化的进化算法，也有普通的局部优化算法；一般选择第一个。

再看一个有等式约束的例子：

$$
\begin{aligned}
f(x) &= (x_1 - 2)^2 + (x_2 - 1)^2 \\
h(x) &= x_1 - 2x_2 + 1 = 0 \\
g(x) &= \tfrac{x_1^2}{4} + x_2^2 - 1 \leq 0
\end{aligned}
$$

```
local print = require('package.print').print
local mol = libminoptlab

function f(x)
    return (x[1] - 2) ^ 2 + (x[2] - 1) ^ 2
end

local lb = {-100, -100}
local ub = {100, 100}

local h = function(x)
    return {x[1] - 2 * x[2] + 1}
end
local g = function(x)
    return {x[1] ^ 2 / 4 + x[2] ^ 2 - 1}
end

local eps = 1e-5
local algo = mol.listoptalgo()[1]
local n = 2000

local best, fbest = mol.fmincon(f, lb, ub, h, g, algo, eps, n)

print({
    best = best,
    fbest = fbest,
    eqcon = h(best),
    ineqcon = g(best)
})
```

7.3 混合整数规划

混合整数规划在 minoptlab 中过于普通，只需在 fmincon 命令的参数序列的最后面添加一个表参数，其元素为整数变量的索引，即可。

比如：

```
// process synthes1
// mixed-integer nonlinear program

// Source: Test problem 1 (Synthesis of processing system) in
// M. Duran & I.E. Grossmann,
// "An outer approximation algorithm for a class of mixed integer nonlinear
//  programs", Mathematical Programming 36, pp. 307-339, 1986.

// Number of variables:   6 (3 binary variables)
// Number of constraints: 6
// Objective nonlinear
// Nonlinear constraints


MODULE: NLP;

BINARY_VARIABLES y1, y2, y3;

POSITIVE_VARIABLES x1, x2, x3;

UPPER_BOUNDS{
x1: 2 ;
x2: 2 ;
x3: 1 ;
}

EQUATIONS c1, c2, c3, c4, c5, c6;

c1: 0.8*log(x2 + 1) + 0.96*log(x1 - x2 + 1) - 0.8*x3 >= 0 ;
c2: log(x2 + 1) + 1.2*log(x1 - x2 + 1) - x3 - 2*y3 >= -2 ;
c3: x2 - x1 <= 0 ;
c4: x2 - 2*y1 <= 0 ;
c5: x1 - x2 - 2*y2 <= 0 ;
```

```
c6: y1 + y2 <= 1 ;

OBJ: minimize
     5*y1 + 6*y2 + 8*y3 + 10*x1 - 7*x3 - 18*log(x2 + 1)
     - 19.2*log(x1 - x2 + 1) + 10;
```

```
local print = require('package.print').print
local mol = libminoptlab

function f(x)
    local x1 = x[1]
    local x2 = x[2]
    local x3 = x[3]
    local y1 = x[4]
    local y2 = x[5]
    local y3 = x[6]
    return 5 * y1 + 6 * y2 + 8 * y3 + 10 * x1 - 7 * x3 - 18 * math.log(x2 + 1) - 19.2 *
    math.log(x1 - x2 + 1) + 10
end

local h = mol.nullconfun()

local g = function(x)
    local x1 = x[1]
    local x2 = x[2]
    local x3 = x[3]
    local y1 = x[4]
    local y2 = x[5]
    local y3 = x[6]
    return {-(0.8 * math.log(x2 + 1) + 0.96 * math.log(x1 - x2 + 1) - 0.8 * x3),
            -(math.log(x2 + 1) + 1.2 * math.log(x1 - x2 + 1) - x3 - 2 * y3) - 2, x2 - x1,
     x2 - 2 * y1, x1 - x2 - 2 * y2,
            y1 + y2 - 1}
end

local lb = {0, 0, 0, 0, 0, 0}
local ub = {2, 2, 1, 1, 1, 1}
local algo = mol.listoptalgo()[1]
local eps = 1e-5
```

```
32 local n = 2000
33 local intindex = {4, 5, 6}
34
35 local best, fbest = mol.fmincon(f, lb, ub, h, g, algo, eps, n, intindex)
36
37 print({
38     best = best,
39     fbest = fbest,
40     ineqcon = g(best)
41 })
```

这种求混合整数规划的方法，对 lpsolve 命令也是有效的。

7.4 多目标

多目标的情况直接用 mxfun 命令将多目标函数转换为单目标函数即可，其余照旧。

比如有两个需要最大化的目标函数：$2x_1+3x_2$ 和 $4x_1-2x_2$，约束条件是两个不大于条件：$x_1+x_2\leq 10$ 和 $2x_1+x_2\leq 15$，并且 $x_1\geq 0$，$x_2\geq 0$。

```
local print = require('package.print').print
local mol = libminoptlab

function f1(x)
    return {-(2 * x[1] + 3 * x[2]), -(4 * x[1] - 2 * x[2])}
end

local f2 = mol.mxfun(f1)

local h = mol.nullconfun()

local g = function(x)
    return {x[1] + x[2] - 10, 2 * x[1] + x[2] - 15}
end

local lb = {0, 0}
local ub = {1e4, 1e4}
local algo = mol.listoptalgo()[1]
local eps = 1e-5
local n = 2000

local best, fbest = mol.fmincon(f2, lb, ub, h, g, algo, eps, n)

print({
    best = best,
    f2 = fbest,
    f1 = f1(best),
    ineqcon = g(best)
})
```

8 最小二乘法

最小二乘法是数据拟合的基本方法。minoptlab 通过最小化残差平方和的方式推算待拟合参数。一般地，数据拟合模型分为两类：线性组合模型

$$y = \sum_{i=1}^{n} p_i f_i(\mathbf{x})$$

和非线性模型

$$y = \sum_{i=1}^{n} p_i f_i(\mathbf{x}, \mathbf{p})$$

前者的一个特例就是一元多项式模型 $y = \sum_{i=1}^{n} p_i x^{i-1}$。所以线性组合模型可视作广义的多项式模型。比如：

$$y = p_1 \sin(x_1) + p_2 \cos(x_2) + p_3 x_1^2 x_2^3$$

对于此类模型，用 `polyfit` 命令进行拟合。其他模型，比如：

$$y = p_1 x_1 \sin(p_2) + p_2 x_2 \cos(p_1) + p_3 x_1^2 x_2^3$$

一律使用 `nlpfit` 命令。

8.1 线性组合模型

假设有一组数据如下：

```
xdat:
0, 0.3000, 0.6000, 0.9000, 1.2000, 1.5000,
1.8000, 2.1000, 2.4000, 2.7000, 3.0000
ydat:
2.0000, 2.3780, 3.9440, 7.3460, 13.2320,22.2500,
35.0480, 52.2740, 74.5760, 102.6020,137.0000
```

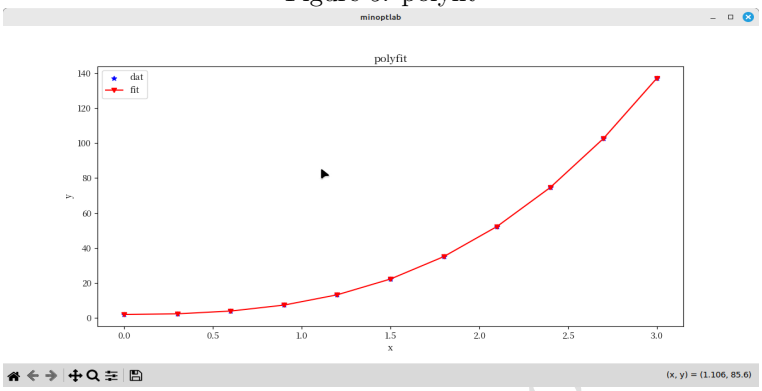
要用一元多项式模型：

$$y = \sum_{i=1}^{4} p_i x^{i-1}$$

拟合这组数据。

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local xdat = mol.matrix.new({0, 0.3000, 0.6000, 0.9000, 1.2000, 1.5000, 1.8000, 2.1000,
    2.4000, 2.7000, 3.0000})
local ydat = {2.0000, 2.3780, 3.9440, 7.3460, 13.2320, 22.2500, 35.0480, 52.2740,
    74.5760, 102.6020, 137.0000}

--[[
function model(x, p)
    return {1, x[1], x[1] ^ 2, x[1] ^ 3}
end
--]]

local model = mol.polyfitfun(4)
local p, corr, fval = mol.polyfit(xdat, ydat, model, 4)

print({
    p = p,
    corr = corr,
    fval = fval
})

local fit = mol.tofit(model, p)
local fity = xdat:apply(fit)

local p1 = plt.scatter({{xdat:tovector(), ydat}}, {'dat'}, {{
    marker = '*',
    color = 'b'
}})
local p2 = plt.line({{xdat:tovector(), fity}}, {'fit'}, {{
    marker = 'v',
    color = 'r'
}})
local fw = plt.framework(111, 'polyfit', {'x', 'y'})
plt.show({{fw, {p1, p2}}})
```

Figure 5: polyfit



一元多项式模型直接用 `polyfitfun` 命令创建，其参数是 p_i 的个数。但也可以自行编写函数。要注意的是，无论 `polyfit` 还是 `nlpfit`，编写模型时要返回数学函数列表，且其元素不应包含代表线性组合含义的待拟合参数部分。也就是说，编写模型函数 $p_i f_i$ 时只应使用 f_i 部分，不能包括 p_i 部分。

`tofit` 命令的作用是把模型和参数组合成解模型。它的第一个参数是模型，第二个参数是参数取值列表。不同的参数取值意味着不同的解模型。不同的解模型意味着不同的相关系数。相关系数是 `polyfit` 和 `nlpfit` 命令的第二个返回值。它的第一个返回值是使得残差平方和最小化的拟合参数，第三个返回值就是最小化的残差平方和。

`matrix` 类的 `apply` 方法能把矩阵的每一行作为参数“投送”给它的唯一参数，这个参数应该是一个多元函数。它会返回一个列表，其元素一一对应于矩阵的每一行。

8.2 非线性模型

`nlpfit` 是拟合非线性模型的通用命令。该功能通过了 NIST（美国国家标准与技术研究院）公布的全部非线性拟合测试。点击 Help↦NIST 中的任一文件名，可加载对应测试脚本并自行完成运行测试。

`nlpfit` 命令内部基于 `fmincon` 命令，所以其参数序列与后者非常相似。第一个参数是 matrix 类表示的自变量数据，第二个参数是用列表表示的因变量数据，第三个参数是模型，与 `polyfit` 的模型参数类似。后面五个参数与 `fmincon` 命令的后面五个参数类似。分别是搜索下界、搜索上界、算法、收敛精度和最大迭代次数。如果有关于参数的等式或者不等式条件约束，应该在算法参数之前输入。

给定一组数据如下：

```
xdat:
0.5,1.5,2.5,3.5,4.5,5.5,6.5,7.5,8.5,9.5,10.5,11.5,12.5,
13.5,14.5,15.5,16.5,17.5,18.5,19.5,20.5,21.5,22.5,23.5,
24.5,25.5,26.5,27.5,28.5,29.5,30.5,31.5,32.5,33.5,34.5
ydat:
0.0202,0.0292,0.0467,0.0658,0.0707,0.0856,0.0832,0.0865,
0.0827,0.0782,0.0677,0.0597,0.0503,0.0401,0.0305,0.0269,
0.0199,0.0133,0.0088,0.0079,0.0082,0.0037,0.0036,0.0026,
0.0015,0.0016,0.0014,0.0011,0.0007,0.0002,0.0006,0.0006,
0.0001,0.0002,0.0001
```

需要拟合的模型是：

$$y = \left(\frac{p_1}{p_2\sqrt{\frac{\pi}{2}}}\right)\exp\left(-2\left(\frac{x_1 - p_3}{p_2}\right)^2\right) + p_4$$

并且要求拟合参数满足以下两个条件：

$$y\left(0.5, p\right) = 0.0202$$
$$y\left(34.5, p\right) = 0.0001$$

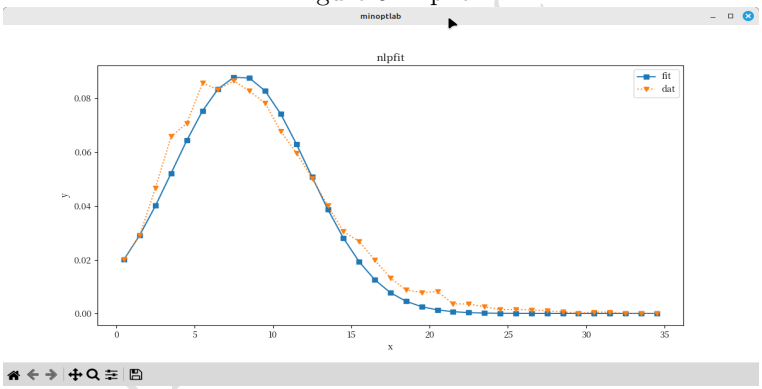
也就是说，拟合曲线必须通过第一个点和最后一个点。

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

```

```
local xdat = mol.matrix.new({0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5, 10.5, 11.5, 12.5, 13.5, 14.5, 15.5, 16.5,
                             17.5, 18.5, 19.5, 20.5, 21.5, 22.5, 23.5, 24.5, 25.5, 26.5, 27.5, 28.5, 29.5, 30.5, 31.5,
                             32.5, 33.5, 34.5})

local ydat = {0.0202, 0.0292, 0.0467, 0.0658, 0.0707, 0.0856, 0.0832, 0.0865, 0.0827, 0.0782, 0.0677, 0.0597, 0.0503,
              0.0401, 0.0305, 0.0269, 0.0199, 0.0133, 0.0088, 0.0079, 0.0082, 0.0037, 0.0036, 0.0026, 0.0015, 0.0016,
              0.0014, 0.0011, 0.0007, 0.0002, 0.0006, 0.0006, 0.0001, 0.0002, 0.0001}

local model = function(x, p)
    return {(1 / (p[2] * math.sqrt(math.pi / 2))) * math.exp(-2 * ((x[1] - p[3]) / p[2]) ^ 2), 0, 0, 1}
end

local eqcon = function(p)
    return {mol.tofit(model, p)({0.5}) - 0.0202, mol.tofit(model, p)({34.5}) - 0.0001}
end

local ineqcon = mol.nullconfun()

local lb = {0, 0, 0, 0}
local ub = {100, 100, 100, 100}
local algo = mol.listoptalgo()[1]
local eps = 1e-5
local n = 2000

local p, corr, fval = mol.nlpfit(xdat, ydat, model, lb, ub, eqcon, ineqcon, algo, eps, n)

print({
    p = p,
    corr = corr,
    fval = fval,
    eqcon = eqcon(p)
})

local fit = mol.tofit(model, p)
```

```
local fity = xdat:apply(fit)

local p1 = plt.line({{xdat:tovector(), fity}}, {'fit'}, {{
    marker = 's'
}})
local p2 = plt.line({{xdat:tovector(), ydat}}, {'dat'}, {{
    marker = 'v',
    linestyle = 'dotted'
}})
local fw = plt.framework(111, 'nlpfit', {'x', 'y'})
plt.show({{fw, {p1, p2}}})
```

Figure 6: nlpfit



9 常微分方程组

ode 类专门用来求解常微分方程组。

比如受重力作用且有摩擦的摆角 θ 的二阶微分方程可以写成:

$$\theta''(t) + b \cdot \theta'(t) + c \cdot \sin(\theta(t)) = 0$$

令:

$$\omega(t) = \theta'(t)$$

则

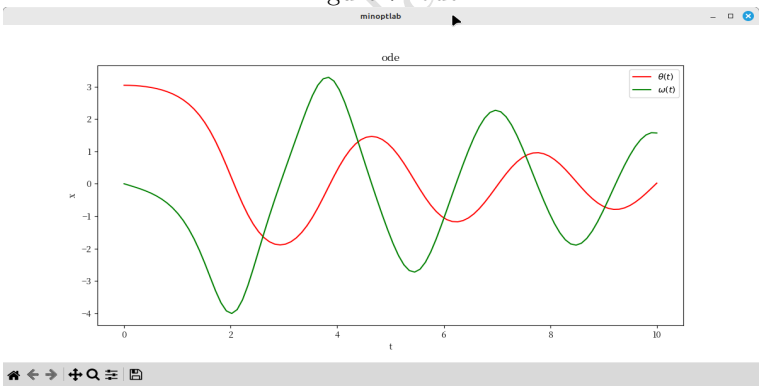
$$\theta'(t) = \omega(t)$$
$$\omega'(t) = -b \cdot \omega(t) - c \cdot \sin(\theta(t))$$

令 $b = 0.25$，$c = 5$。对于初始条件，我们假设摆几乎垂直，$\theta(0) = \pi - 0.1$，并且初始时处于静止状态，因此 $\omega(0) = 0$。我们将在 $0 \leq t \leq 10$ 的区间内，在 100 个等间隔的样本点生成解决方案:

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local b, c = 0.25, 5.0

function system(x,t)
    return {x[2], -b * x[2] - c * math.sin(x[1])}
end

local x0 = {math.pi - 0.1, 0}
local t0, t1 = 0, 10
local step = 0.1


local odesolver = mol.ode.new(system, x0, t0, t1, step, false);

local t = mol.linspace(t0, t1, 100)
local x = mol.matrix.new(#t, 2)
```

```
20
21 for i = 1, #t do
22     x:set(i, 1, odesolver:get(1, t[i]))
23     x:set(i, 2, odesolver:get(2, t[i]))
24 end
25
26
27 local p1 = plt.line({{t, x:col(1):tovector()}}, {'$\\theta(t)$'}, {{
28     color = 'r'
29 }})
30 local p2 = plt.line({{t, x:col(2):tovector()}}, {'$\\omega(t)$'}, {{
31     color = 'g'
32 }})
33 local fw = plt.framework(111, 'ode', {'t', 'x'})
34 plt.show({{fw, {p1, p2}}})
```

Figure 7: ode



ode 类的构造方法的参数分别是方程系统、初值、求解范围起点、终点、步长和是否刚性。get 方法负责给出对应于给定点的解函数的值，它有两个参数：第一个参数指定哪个解函数，第二个参数指定点位置。

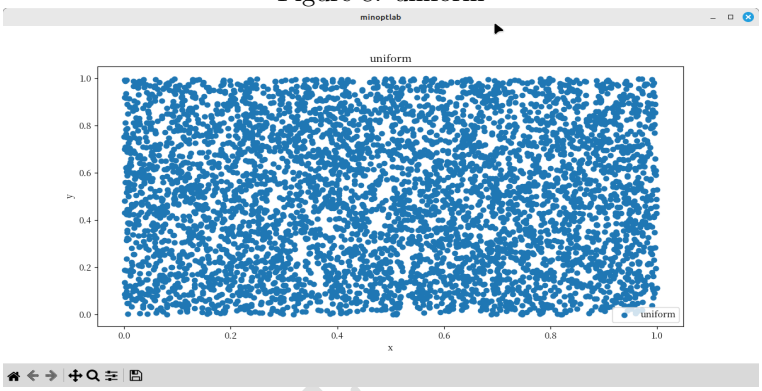
要注意的是，刚性方程组一定要设置 ode 类构造方法的最后参数为 true，这将大大有利于方程组的求解。

10 随机数与统计

10.1 随机数

`uniform` 类负责提供 0 到 1 之间的均匀随机数。

Figure 8: uniform




```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local rnd = mol.uniform.new()

local x = {}
local y = {}

local n = 5000

for i = 1 ,n do
    table.insert(x,rnd:generate())
    table.insert(y,rnd:generate())
end

local img = plt.scatter({{x,y}},{'uniform'})
local fw = plt.framework(111,'uniform',{'x','y'})
```

```
plt.show({{fw,{img}}})
```

generate 方法每次输出一个 $a_i \in [0,1]$。

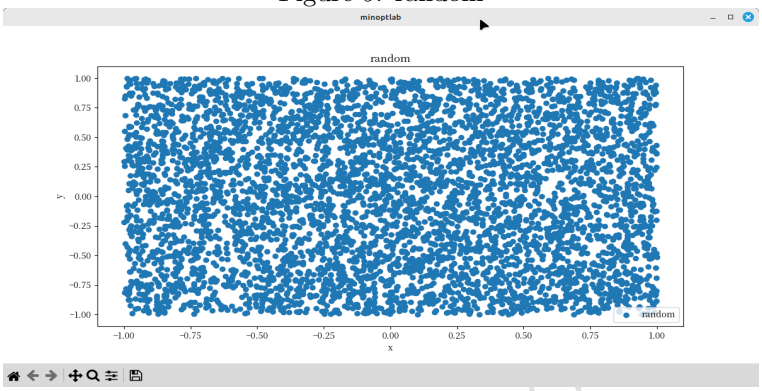
如果需要其他区间的均匀随机数，可利用公式：$b_i = l + a_i(u-l)$。l 表示下界，u 表示上界。

另一种方法是直接使用 random 类。它的构造方法有两个参数，分别是 l 和 u：

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local rnd = mol.random.new(-1,1)

local x = {}
local y = {}

local n = 5000

for i = 1 ,n do
    table.insert(x,rnd:generate())
    table.insert(y,rnd:generate())
end

local img = plt.scatter({{x,y}},{'random'})
local fw = plt.framework(111,'random',{'x','y'})
plt.show({{fw,{img}}})
```

random 类也有一个 generate 方法。

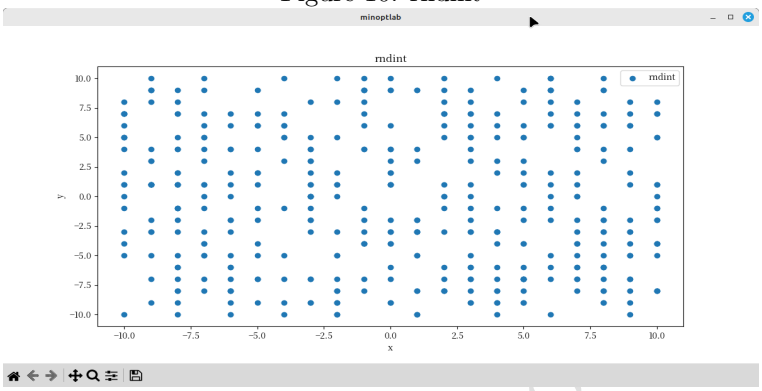
Figure 9: random



如果需要指定区间的随机整数，可以使用另一个 `random` 类。这个 `random` 类不在 libminoptlab 名下，可以直接使用。

```
local print = require('package.print').print
local plt = require('package.plot')

local rndint = random.new(os.time())

local l, u = -10, 10
local n = 400
local x = {}
local y = {}

for i = 1, n do
    table.insert(x, rndint:value(l, u))
    table.insert(y, rndint:value(l, u))
end

local img = plt.scatter({{x, y}}, {'rndint'})
local fw = plt.framework(111, 'rndint', {'x', 'y'})
plt.show({{fw, {img}}})
```

Figure 10: rndint



它的 `value` 方法有两个参数，分别指定生成数的下界和上界。若调用该方法时不输入任何参数，则其生成数为位于 0 至 1 之间的均匀随机实数。

10.2 统计

minoptlab 内置用于统计计算的工具箱，详见下表：

sum	和	var	方差
mul	积	cov	协方差
mean	平均值	cor	相关系数
skewness	偏度	{d,p,q,r}[2]bern	Bernoulli distribution
kurtosis	峰度	{d,p,q,r}cauchy	Cauchy distribution
first_four_moments	前四个矩	{d,p,q,r}beta	Beta Distribution
median	中位数	{d,p,q,r}binom	Binomial Distribution
median_absolute_deviation	中位数绝对偏差	{d,p,q,r}chisq	Chi-squared distribution
interquartile_range	四分位距	{d,p,q,r}exp	Exponential Distribution
gini_coefficient	基尼系数	{d,p,q,r}f	F-Distribution
mode	众数	{d,p,q,r}gamma	Gamma Distribution
{d,p,q,r}invgamma	Inverse-Gamma Distribution	{d,p,q,r}invgauss	Inverse-Gaussian Distribution
{d,p,q,r}laplace	Laplace Distribution	{d,p,q,r}lnorm	Log-Normal Distribution
{d,p,q,r}logis	Logistic Distribution	{d,p,q,r}norm	Normal Distribution
{d,p,q,r}pois	Poisson Distribution	{d,p,q,r}radem	Rademacher Distribution
{d,p,q,r}t	Student’ s t-Distribution	{d,p,q,r}unif	Uniform Distribution
{d,p,q,r}weibull	Weibull Distribution		

[2]d，p，q，r 分别代表 density function，distribution function，quantile function 和 random sampling function。

11 傅立叶变换

minoptlab 提供一维和二维的傅立叶变换及其逆操作。

如果变换的命令是 xyz，那么逆变换的命令就是 ixyz。

11.1 一维的情况

如果只需要傅立叶变换的实部信息，那么可以用 r2r 和 ir2r 命令。前者返回实部信息，后者使用这个信息返回原数据。

```
local print = require('package.print').print
local mol = libminoptlab

local x = mol.linspace(0, 1, 10)
local y = mol.r2r(x)
local z = mol.ir2r(y)
print({
    x = x,
    y = y,
    z = z,
    eps = mol.vector.new(x):sub(z):norm()
})

-------------------------------------------------------

local yr, yi = mol.r2c(x)
z = mol.ir2c(yr, yi)
print({
    x = x,
    yr = yr,
    yi = yi,
    z = z,
    eps = mol.vector.new(x):sub(z):norm()
})

-------------------------------------------------------

local xr, xi = {0, 1, 10}, {3, 5, 8}
```

```
yr, yi = mol.c2c(xr, xi)
local zr, zi = mol.ic2c(yr, yi)
print({
    xr = xr,
    xi = xi,
    yr = yr,
    yi = yi,
    zr = zr,
    zi = zi,
    eps = {mol.vector.new(xr):sub(zr):norm(), mol.vector.new(xi):sub(zi):norm()}
})
```

若关心虚部信息，则可以用 r2c 和 ir2c 命令。前者返回实部和虚部信息，后者使用这两个信息返回原数据。

若原数据包含虚部信息，则应该用 c2c 和 ic2c 命令。前者返回实部和虚部信息，后者使用这两个信息返回原数据。

成对的变换和逆变换命令不是混用。

vector 类是 matrix 类的子类，即仅有一列的 matrix 类。vector 比列表要快，但是没有列表灵活和通用。

11.2 二维的情况

r2r 和 ir2r 命令不支持二维。其他则可以。

所有支持二维的命令都需要在原来一维命令的参数前面添加两个命令：数据的行数和列数。

```
local print = require('package.print').print
local mol = libminoptlab

local nr, nc = 3, 3
local x = mol.matrix.new(nr, nc, {2, 3, 1, 1, 2, 2, 2, 1, 3})
print(x)

local yr, yi = mol.r2c(nr, nc, x:tovector())
print(mol.matrix.new(nr, nc, yr))
print(mol.matrix.new(nr, nc, yi))

local z = mol.ir2c(nr, nc, yr, yi)
print(mol.matrix.new(nr, nc, z))
print(x:sub(z):norm())

------------------------------------------------------------

local xr, xi = x, mol.matrix.rand(nr, nc)
print(xi)

yr, yi = mol.c2c(nr, nc, xr:tovector(), xi:tovector())
print(mol.matrix.new(nr, nc, yr))
print(mol.matrix.new(nr, nc, yi))

local zr, zi = mol.ic2c(nr, nc, yr, yi)
print(mol.matrix.new(nr, nc, zr))
print(mol.matrix.new(nr, nc, zi))

print({
    eps = {xr:sub(zr):norm(), xi:sub(zi):norm()}
})
```

12 矩阵分解

线性代数功能主要是通过 matrix 类完成的。这个类包含的方法基本覆盖了常见的矩阵操作。详见下表:

nrow	行数（属性）	rand	随机矩阵（静态方法）	trace	迹	row	访问行
ncol	列数（属性）	constant	常数矩阵（静态方法）	eig	特征值分解	col	访问列
mul	矩阵乘积	ones	全 1 矩阵（静态方法）	qr	qr 分解	block	访问块
add	加	eye	单位矩阵（静态方法）	lu	lu 分解	tovector	转换成向量表
sub	减	linspace	从指定区间生成均分矩阵（静态方法）	chol	chol 分解	view	转换成矩阵表
prod	乘	meshgridXY	二维网格数据矩阵（静态方法）	svd	svd 分解	reshape	重塑矩阵
div	除	meshgridXYZ	三维网格数据矩阵（静态方法）	schur	schur 分解	min	最小值
get	元素访问	inv	方阵求逆	hessenberg	hessenberg 分解	max	最大值
set	元素设置	pinv	伪逆	adjoint	伴随矩阵	read	从 csv 文件读入数据
apply	逐行投喂给指定函数	transpose	转置	diagonal	对角矩阵	write	把数据写入 csv 文件
norm	2-范数	dot	点积	asdiagonal	生成对角矩阵	cross	叉积
foreach	逐个投喂给指定函数	det	行列式	triangular	三角矩阵	tableview	转换成表型字符串
fill	填充	rank	秩	listtriangular	三角矩阵属性列表	solve	解线性方程组
sum	逐行求和	mean	逐行求均值				

Figure 11: 矩阵操作

Figure 12: 矩阵操作部分说明

	参数数量	参数序列	功能	性质
nrow			行数	属性
ncol			列数	属性
mul	1	矩阵或者列表	矩阵乘法，返回矩阵	方法
add	1	矩阵、列表或者标量	向量化加法，返回矩阵	方法
sub	1	矩阵、列表或者标量	向量化减法，返回矩阵	方法
prod	1	矩阵、列表或者标量	向量化乘法，返回矩阵	方法
prod	0		按列累乘，返回矩阵	方法
div	1	矩阵、列表或者标量	向量化除法，返回矩阵	方法
get	1	非负整数	向量式访问元素，返回实数	方法
get	2	非负整数	矩阵式访问元素，返回实数	方法
set	2	非负整数、实数	向量式设置元素，无返回	方法
set	3	非负整数、非负整数、实数	矩阵式设置元素，无返回	方法
apply	1	函数	逐行调用函数，返回矩阵	方法
foreach	1	函数	逐个调用函数，修改矩阵，无返回	方法
sum	0		逐行求和，返回矩阵	方法
mean	0		逐行求均值，返回矩阵	方法
listtriangular	0		返回 triangular 方法可用参数列表	方法
triangular	1	listtriangular 返回值中的一个	取三角矩阵	方法

12.1 特征值分解

$$A = VDV^{-1}$$

```
1 local print = require('package.print').print
2 local plt = require('package.plot')
3 local mol = libminoptlab
4
```

```
local m = mol.matrix.rand(3,3)
local eig = m:eig()
local eigvalues = eig[1]
local eigvectors = eig[2]
local check = m:mul(eigvectors):sub(eigvectors:mul(eigvalues))

print(m:tableview('m'))
print(eigvalues:tableview("eigvalues"))
print(eigvectors:tableview('eigvectors'))
print(check:tableview('check'))
```

12.2 lu 分解

$$A = P^{-1}LUQ^{-1}$$

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local m = mol.matrix.rand(3,3)
local lu = m:lu()
local p = lu[1]
local l = lu[2]
local u = lu[3]
local q = lu[4]
local check = m:sub(p:inv():mul(l):mul(u):mul(q:inv()))

print(m:tableview('m'))
print(p:tableview("p"))
print(l:tableview("l"))
print(u:tableview("u"))
print(q:tableview("q"))
print(check:tableview('check'))
```

12.3 qr 分解

$$A = QR$$

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local m = mol.matrix.rand(3,3)
local qr = m:qr()
local q = qr[1]
local r = qr[2]
local check = m:sub(q:mul(r))

print(m:tableview('m'))
print(q:tableview("q"))
print(r:tableview("r"))
print(check:tableview('check'))
```

12.4 chol 分解

$$A = LL^T = U^T U$$

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local m = mol.matrix.new(3,3,{4,-1,2, -1,6,0, 2,0,5})
local chol = m:chol()
local l = chol[1]
local u = chol[2]
local check1 = m:sub(l:mul(l:transpose()))
local check2 = m:sub(u:transpose():mul(u))

print(m:tableview('m'))
print(l:tableview("l"))
print(u:tableview("u"))
print(check1:tableview('check1'))
print(check2:tableview('check2'))
```

12.5 svd 分解

$$A = USV^T$$

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local m = mol.matrix.rand(3,3)
local svd = m:svd()
local u = svd[1]
local s = svd[2]
local S = s:asdiagonal(m.nrow,m.ncol)
local v = svd[3]

local check = m:sub(u:mul(S):mul(v:transpose()))

print(m:tableview('m'))
print(u:tableview("u"))
print(s:tableview("s"))
print(S:tableview("S"))
print(v:tableview("v"))
print(check:tableview('check'))
```

12.6 schur 分解

$$A = UTU^T$$

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local m = mol.matrix.rand(3,3)
local svd = m:svd()
local u = svd[1]
local s = svd[2]
local S = s:asdiagonal(m.nrow,m.ncol)
local v = svd[3]

local check = m:sub(u:mul(S):mul(v:transpose()))

print(m:tableview('m'))
print(u:tableview("u"))
print(s:tableview("s"))
print(S:tableview("S"))
print(v:tableview("v"))
print(check:tableview('check'))
```

12.7 hessenberg 分解

$$A = QHQ^T$$

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local m = mol.matrix.rand(3,3)
local hessenberg = m:hessenberg()
local h = hessenberg[1]
local q = hessenberg[2]

local check = m:sub(q:mul(h):mul(q:transpose()))

print(m:tableview('m'))
print(h:tableview("h"))
print(q:tableview("q"))
print(check:tableview('check'))
```

13 绘图

绘图功能的核心概念有三个：构图、标架、显示。所有 2D 和 3D 的绘图都围绕着这三个概念展开。

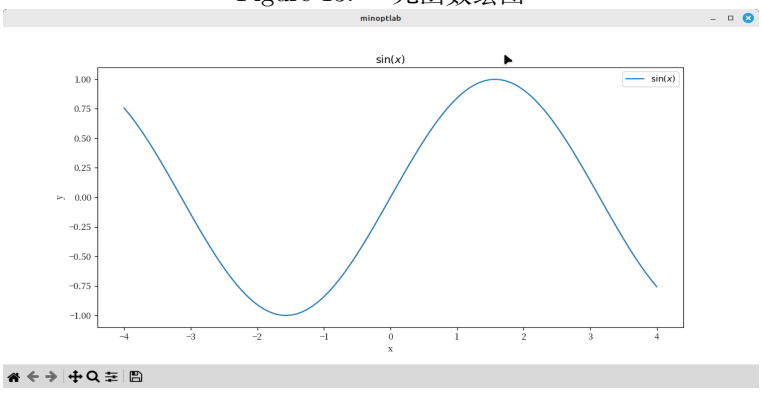
	构图	标架	显示
二维	`fplot、line、scatter、apply`	`framework`	`show`
三维	`fplot3d、line3d、scatter3d、contour、apply3d、plot3d`	`frameword3d`	

13.1 一元函数

一元函数绘图用 `fplot` 命令。

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local f = function(x)
    return math.sin(x)
end

local img = plt.fplot({f},{{-4}},{{4}},{'$\\sin(x)$'})
local fw = plt.framework(111,'$\\sin(x)$',{'x','y'})
plt.show({{fw,{img}}})
```

Figure 13: 一元函数绘图



`fplot` 命令的参数序列是：函数、下界、上界、标签、采样点数、样式、扩展。后三个为可选项。每一个参数都要用列表来表示。

13.2 二维数据

二维数据绘图可用 line、scatter 和 apply。apply 能选择的类型包括：plot、bar、stem、pie、step、hist 等。

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local x = mol.linspace(-5, 5, 50)
local y = mol.apply(x, math.sin)

local images = {}

local line = plt.line({{x, y}},{'$\\sin(x)$'})
local fw1 = plt.framework(241, 'line', {'x', 'y'})
table.insert(images, {fw1, {line}})

local scatter = plt.scatter({{x, y}}, {'$\\sin(x)$'}, {{
    color = 'r',
    marker = '*'
}})
local fw2 = plt.framework(242, 'scatter', {'x', 'y'})
table.insert(images, {fw2, {scatter}})

local a = {1, 2, 3, 4}
local b = mol.linspace(1, 10, 4)
local bar = plt.apply('bar', {{a, b}}, {'sample'})
local fw3 = plt.framework(243, 'bar', {'x', 'y'})
table.insert(images, {fw3, {bar}})

local stem = plt.apply('stem', {{x, y}}, {'$\\sin(x)$'})
local fw4 = plt.framework(244, 'stem', {'x', 'y'})
table.insert(images, {fw4, {stem}})

local step = plt.apply('step', {{x, y}}, {'$\\sin(x)$'}, {{
    color = 'g'
}})
local fw5 = plt.framework(245, 'step', {'x', 'y'})
```

```
35 table.insert(images, {fw5, {step}})
36
37 local c = mol.rnorm(1000, 1, 0, 1):tovector()
38 local hist = plt.apply('hist', {{c}}, {'sample'}, {{
39     color = 'skyblue',
40     edgecolor = 'black',
41     bins = 30
42 }})
43 local fw6 = plt.framework(246, 'hist', {'x', 'y'})
44 table.insert(images, {fw6, {hist}})
45
46 local fruits = {'apple', 'blueberry', 'cherry', 'orange'}
47 local counts = {40, 60, 30, 55}
48 local pie = plt.apply('pie', {{counts}}, {}, {{
49     labels = fruits
50 }})
51 local fw7 = plt.framework(247, 'pie', {})
52 table.insert(images, {fw7, {pie}})
53
54 local polar = plt.apply('plot', {{x, y}}, {'$\\sin(x)$'})
55 local fw8 = plt.framework(248, 'polar', {'x', 'y'}, {
56     projection = 'polar'
57 })
58 table.insert(images, {fw8, {polar}})
59
60 plt.show(images)
```

Figure 14: 二维数据

`apply` 命令的参数序列是类型、数据、标签、样式和扩展。后两项是可选的。每一个参数都要用列表来表示。

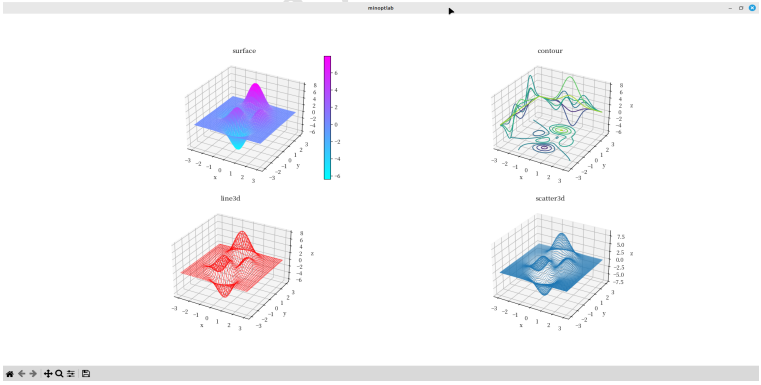
13.3 二元函数

绘制二元函数的图像，可以直接 `fplot3d` 命令，使用 `line3d` 和 `scatter3d` 也是可以的。contour 和 contourf 命令则可以绘制等高图。

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local peaks = function(xx)
    local x, y = xx[1], xx[2]
    return
        3 * (1 - x) ^ 2 * math.exp(-(x ^ 2) - (y + 1) ^ 2) - 10 * (x / 5 - x ^ 3 - y ^ 5)
     * math.exp(-x ^ 2 - y ^ 2) - 1 /
            3 * math.exp(-(x + 1) ^ 2 - y ^ 2)
end

local lb = {-3, -3}
local ub = {3, 3}
local n = 80

images = {}

local p1 = plt.fplot3d({peaks}, {lb}, {ub}, n, {{
    cmap = 'cool'
}})
local fw1 = plt.framework3d(221, 'surface', {'x', 'y', 'z'})
table.insert(images, {fw1, {p1}})

local p2 = plt.contour({peaks, peaks, peaks}, {lb, lb, lb}, {ub, ub, ub}, n, {{
    zdir = 'x',
    offset = -3
}, {
    zdir = 'y',
    offset = 3
}, {
    zdir = 'z',
    offset = -6
}})
```

```
local fw2 = plt.framework3d(222, 'contour', {'x', 'y', 'z'})
table.insert(images, {fw2, {p2}})

local p3 = plt.line3d({peaks}, {lb}, {ub}, n, {{
    color = 'r',
    linewidth = 0.5
}})
local fw3 = plt.framework3d(223, 'line3d', {'x', 'y', 'z'})
table.insert(images, {fw3, {p3}})

local p4 = plt.scatter3d({peaks}, {lb}, {ub}, n, {{
    s = 0.5
}})
local fw4 = plt.framework3d(224, 'scatter3d', {'x', 'y', 'z'})
table.insert(images, {fw4, {p4}})

plt.show(images)
```

Figure 15: 二元函数绘图



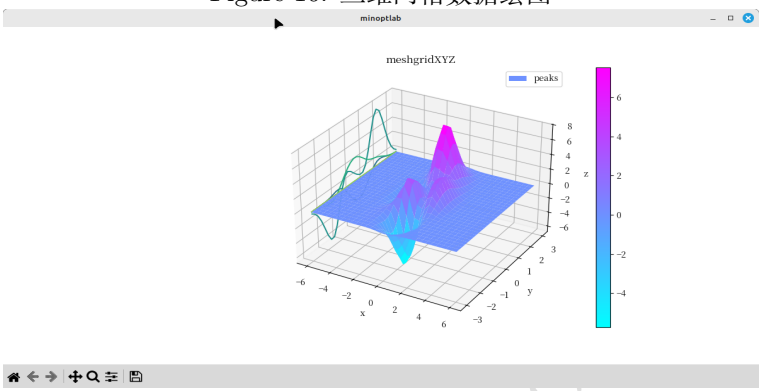
`fplot3d`、`line3d`、`scatter`、`contour` 和 `contourf` 命令的参数序列是一致的，分别是函数、下界、上界、采样数、样式和扩展。后三个参数是可选的。每一个参数都必须用列表表示。

13.4 三维网格数据

三维网格数据用 matrix 类的静态方法 meshgridXYZ 来创建。然后用 plot3d 命令创建图形。

```
local print = require('package.print').print
local plt = require('package.plot')
local mol = libminoptlab

local peaks = function(xx)
    local x, y = xx[1], xx[2]
    return
        3 * (1 - x) ^ 2 * math.exp(-(x ^ 2) - (y + 1) ^ 2) - 10 * (x / 5 - x ^ 3 - y ^ 5)
     * math.exp(-x ^ 2 - y ^ 2) - 1 /
           3 * math.exp(-(x + 1) ^ 2 - y ^ 2)
end

local x = mol.linspace(-6, 6, 30)
local y = mol.linspace(-3, 3, 30)

local xyz = mol.matrix.meshgridXYZ(x, y, peaks)
local xdat, ydat, zdat = xyz[1]:view(), xyz[2]:view(), xyz[3]:view()

local img1 = plt.plot3d('plot_surface', {{xdat, ydat, zdat}}, {'peaks'}, {{
    cmap = 'cool'
}})
local img2 = plt.plot3d('contour', {{xdat, ydat, zdat}}, {'contour'}, {{
    zdir = 'x',
    offset = -6
}})

local fw = plt.framework3d(111, 'meshgridXYZ', {'x', 'y', 'z'})
plt.show({{fw, {img1, img2}}})
```

Figure 16: 三维网格数据绘图



`meshgridXYZ` 命令的三个参数分别是第一、二维自变量数据列表和二元采样函数。它返回包含三个 `matrix` 元素的列表，分别对应于三个维度的采样数据。`plot3d` 命令利用这些采样数据完成绘图工作。对于 `meshgridXYZ` 生成的二维数据，plot3d 的第一个参数可选如下值：`plot_surface`，`plot_wireframe`，`contourf`，`contour`，scatter。如果是 `linspace` 生成的一维数据，则可选的值有：`plot`，`scatter`。