

AnyProg
C++ 数学优化包

`pangpang@hi-nginx.com`

2019 年 11 月 30 日

目录

| | | |
|----------|-------------------------|-----------|
| 1 | 起源 | 5 |
| 2 | 功能 | 7 |
| 3 | 技巧 | 9 |
| 3.1 | 线性优化 | 9 |
| 3.2 | 二次优化 | 10 |
| 3.3 | 非线性优化 | 11 |
| 3.4 | 整数优化 | 13 |
| 3.5 | TSP 问题 | 14 |
| 3.6 | Assignment 问题 | 14 |
| 3.7 | 数据拟合问题 | 14 |
| 3.8 | 方程组求解 | 14 |
| 4 | 展望 | 15 |

Chapter 1

起源

应该有一个 C++ 版的 LINGO。

它必须免费、高效而且易用。

它必须以 C++ 的方式解决 LINGO 需要面对的问题。

它的名字是：AnyProg¹。

¹Any Programming。

Chapter 2

功能

AnyProg 可用于求解以下模型定义的所有问题:

$$\begin{aligned} \min_{x \in \mathbf{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, \quad i = 0, \dots, m_s - 1 \\ & h_j(x) = 0, \quad j = 0, \dots, m_t - 1 \\ & x_k \in [L_k, U_k], \quad k = 0, \dots, n - 1 \end{aligned} \tag{2.1}$$

其中, f 是目标函数, g_i 和 h_j 分别是不等式约束函数 (m_s 个) 和等式约束函数 (m_t 个), x 是一个 n 维实变量, 其分量 x_k 位于闭区间 $[L_k, U_k]$ 。模型 (2.1) 不仅适合于连续优化问题, 也适合于离散优化问题, 还适合于混合优化问题。对 AnyProg 来说, 更重要的概念是解的全局性, 而不是变量的连续性。同时, 目标函数和约束函数是否线性并不重要, AnyProg 并不区别看待它们。

AnyProg 能处理的问题的类型十分广泛: **LP**¹、**QP**²、**NLP**³、**MILP**⁴、**MIQP**⁵、**MINLP**⁶。此外, 它也能进行数据的非线性最小二乘拟合, 以及非线性方程组的求解。

¹线性优化

²二次优化

³非线性优化

⁴混合整数线性优化

⁵混合整数二次优化

⁶混合整数非线性优化

Chapter 3

技巧

3.1 线性优化

线性优化的模型定义最好采用矩阵方式，类似于 MATLAB。如例 (3.1)：

$$\begin{aligned} \min_{x \in \mathbf{R}^2} \quad & f(x) = 8x_0 + x_1 \\ \text{s.t.} \quad & g_0(x) = x_0 + 2x_1 \geq -14 \\ & g_1(x) = -4x_0 - x_1 \leq -33 \\ & g_2(x) = 2x_0 + x_1 \leq 20 \\ & x_i \geq 0, \quad i = 0, 1 \end{aligned} \tag{3.1}$$

对于不等于约束，如果采用的是 \geq 方式，则两边乘以 -1 使之反转。如此，原问题可用矩阵表示如下：

$$obj = \begin{pmatrix} 8 \\ 1 \end{pmatrix}, A = \begin{pmatrix} -1 & -2 \\ -4 & -1 \\ 2 & 1 \end{pmatrix}, b = \begin{pmatrix} 14 \\ -33 \\ 20 \end{pmatrix}.$$

在 AnyProg 中，类 `real_block` 用来表示矩阵。因此，例 (3.1) 模型定义如下：

```
1  #include <anyprog/anyprog.hpp>
2
3  int main(int argc, char** argv)
4  {
5      size_t dim = 2;
6      anyprog::real_block obj(dim, 1), A(3, dim), b(3, 1);
7      obj << 8, 1;
8      A << -1, -2,
9          -4, -1,
10         2, 1;
11      b << 14, -33, 20;
12
13      anyprog::optimization::range_t range = { 0, 100 };
14
15      anyprog::optimization opt(obj, range);
```

```

16  opt.set_inequation_condition(A, b);
17  auto ret = opt.solve();
18  anyprog::optimization::print(opt.is_ok(), ret, obj);
19  return 0;
20  }

```

编译、运行后可知，解是 $f(6.5, 7) = 59$ 。本例不包含等式约束。若有，则可效法 A 、 b ，炮制 Aeq 和 beq ，然后使用方法 `set_equation_condition(Aeq, beq)` 添加等式约束，继而求解。本例中的 `range` 是范围，表示 $\forall i, x_i \in [0, 100]$ 。若变量的不同分量有不同的范围限制，可用容器 `std::vector` 包装范围类 `range_t` 列表，并按顺序定义各分量范围。若有较好初值，则可替换范围参数。使用范围参数意味着初值是随机的，有可能不能求解。此时可多次运行程序。随机初值引起的解的不确定性问题。解决该问题的最佳方法是用 `search` 方法取代 `solve` 方法；前者对应于全局解，后者对应于局部解。使用该方法时，对于局部解较少的模型，务必减少前两个参数¹的数值。

3.2 二次优化

二次优化的模型定义如下：

$$\begin{aligned}
 \min_{x \in \mathbf{R}^n} \quad & f(x) = \frac{1}{2}x^T Hx + c^T x \\
 \text{s.t.} \quad & A \cdot x \leq b \\
 & Aeq \cdot x = beq \\
 & x_k \in [L_k, U_k], \quad k = 0, \dots, n-1
 \end{aligned} \tag{3.2}$$

该模型用矩阵形式表示。在求解该类模型时，最好的方式就是顺其自然。比如下例：

$$\begin{aligned}
 \min_{x \in \mathbf{R}^2} \quad & f(x) = \frac{1}{2}x_0^2 + x_1^2 - x_0x_1 - 2x_0 - 6x_1 \\
 \text{s.t.} \quad & g_0(x) = x_0 + x_1 \leq 2 \\
 & g_1(x) = -x_0 + 2x_1 \leq 2 \\
 & g_2(x) = 2x_0 + x_1 \leq 3 \\
 & x_i \geq 0, \quad i = 0, 1
 \end{aligned} \tag{3.3}$$

其矩阵表示如下：

$$H = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}, A = \begin{pmatrix} 1 & 1 \\ -1 & 2 \\ 2 & 1 \end{pmatrix}, b = \begin{pmatrix} 2 \\ 2 \\ 3 \end{pmatrix}, c = \begin{pmatrix} -2 \\ -6 \end{pmatrix}$$

因此，它的模型定义为：

```

1  #include <anyprog/anyprog.hpp>
2
3  int main(int argc, char** argv)
4  {

```

¹`search` 方法的前两个参数的默认值分别是 100 和 30，这是专门应对变态测试模型的基本配置。对于局部解较少的普通模型，两个数值都可以大幅度降低；一般设置 10 和 3 即可。

```

5     size_t dim = 2;
6     anyprog::real_block H(dim, dim), c(dim, 1);
7     H << 1, -1,
8         -1, 2;
9     c << -2, -6;
10
11     anyprog::optimization::function_t obj = [&](const anyprog::real_block& x) {
12         anyprog::real_block ret = 0.5 * (x.transpose() * H * x) + c.transpose() * x;
13         return ret(0, 0);
14     };
15
16     anyprog::real_block A(3, dim), b(3, 1);
17     A << 1, 1,
18         -1, 2,
19         2, 1;
20     b << 2, 2, 3;
21
22     anyprog::optimization::range_t range = { 0, 100 };
23
24     anyprog::optimization opt(obj, range, dim);
25     opt.set_inequation_condition(A, b);
26     auto ret = opt.solve();
27     anyprog::optimization::print(opt.is_ok(), ret, obj);
28     return 0;
29 }

```

编译、运行，其解为 $f(0.666667, 1.33333) = -8.22222$

3.3 非线性优化

非线性优化是最直观地匹配模型2.1的优化类型。如下例 (3.4):

$$\begin{aligned}
 \max_{x \in \mathbf{R}^2} \quad & f(x) = \log x_0 + \log x_1 \\
 \text{s.t.} \quad & g_0(x) = x_0 - x_1 \leq 0 \\
 & h_0(x) = x_0 + 2x_1 = 5 \\
 & x_i \geq 0, \quad i = 0, 1
 \end{aligned} \tag{3.4}$$

这个例子即有不等式约束，也有等式约束。同时，该模型是求最大值而非最小值，可通过目标函数乘以 -1 来转换为标准形式。如此，该模型可定义如下：

```

1  #include <anyprog/anyprog.hpp>
2
3  int main(int argc, char** argv)
4  {
5      size_t dim = 2;
6      anyprog::optimization::function_t obj = [](const anyprog::real_block& x) {
7          return -log(x(0)) - log(x(1));
8      };
9

```

```

10  std::vector<anyprog::optimization::inequation_condition_function_t> ineq = {
11      [&](const anyprog::real_block& x) {
12          return x(0) - x(1);
13      }
14  };
15
16  std::vector<anyprog::optimization::equation_condition_function_t> eq = {
17      [&](const anyprog::real_block& x) {
18          return x(0) + 2 * x(1) - 5;
19      }
20  };
21
22  anyprog::optimization::range_t range = { 0, 10 };
23
24  anyprog::optimization opt(obj, range, dim);
25  opt.set_inequation_condition(ineq);
26  opt.set_equation_condition(eq);
27  auto ret = opt.solve();
28  anyprog::optimization::print(opt.is_ok(), ret, obj);
29  return 0;
30 }

```

编译、运行获得解： $f(1.66667, 1.66667) = 1.02165$ 。例子 (3.4) 或许过于简单了。然而，无论目标函数和约束函数取何种形式，从 AnyProg 的角度来说，它们都是无差异的——重要的是描述模型的结构，而非模型的具体构造。

有时，人们希望在求解时使用函数梯度信息。其实，这并非必须。实际上，这常常费力不讨好。

对于无约束的情况，也可以通过 AnyProg 提供两个静态方法 `fminbnd` 和 `fminunc` 进行求解。假设需要求解 Rosenbrock² 函数 $f(x) = 100(x_1 - x_0^2)^2 + (1 - x_0)^2$ 的最小值，变量范围限制为 $\forall i, x \in [-10, 10]$ 。那么，求解程序会很简单：

```

1  #include <anyprog/anyprog.hpp>
2
3  int main(int argc, char** argv)
4  {
5      size_t dim = 2;
6      anyprog::optimization::function_t obj = [](const anyprog::real_block& x) {
7          return 100 * pow(x(1) - pow(x(0), 2), 2) + pow(1 - x(0), 2);
8      };
9
10     anyprog::optimization::range_t range = { -10, 10 };
11
12     auto ret = anyprog::optimization::fminbnd(obj, range, dim, 1e-10);
13     anyprog::optimization::print(true, ret, obj);
14
15     return 0;
16 }

```

当然，若有较理想初值，也可使用 `fminunc` 求解：

²该函数的全局解 $x = (1, 1)^T$ ，对应的函数值是 0。请参考：<http://mathworld.wolfram.com/RosenbrockFunction.html>。

```

1  #include <anyprog/anyprog.hpp>
2
3  int main(int argc, char** argv)
4  {
5      size_t dim = 2;
6      anyprog::optimization::function_t obj = [](const anyprog::real_block& x) {
7          return 100 * pow(x(1, 0) - pow(x(0, 0), 2), 2) + pow(1 - x(0, 0), 2);
8      };
9
10     anyprog::real_block param(dim, 1);
11     param.fill(0);
12
13     auto ret = anyprog::optimization::fminunc(obj, param, 1e-10);
14     anyprog::optimization::print(true, ret, obj);
15
16     return 0;
17 }

```

3.4 整数优化

整数优化不仅包括纯整数优化，也包括混合整数优化。所谓整数，既指一般整数的情况： $\exists i x_i \in \mathbb{Z}$ ，也指 $0 \mid 1$ 值的情况： $\exists i, x_i \in \{0, 1\}$ 。

整数优化通过两个途径达到：第一、调用 `set_enable_integer_filter` 或者 `set_enable_binary_filter` 方法，第二、调用 `set_filter_function` 方法。前者适合于纯整数优化，后者则适合混合整数优化。此二途径对所有符合模型 (2.1) 的优化问题均有效。`set_enable_integer_filter` 方法保证解的每一分量皆为限制范围内的整数，`set_enable_binary_filter` 方法则保证解的每一分量不是 0 便是 1。

来看例子 (3.5)：

$$\begin{aligned}
 \min_{x \in \mathbf{R}^2} \quad & f(x) = 0.5x_0^2 + 0.5x_1^2 - 2x_0 - 2x_1 \\
 \text{s.t.} \quad & g_0(x) = -x_0 + x_1 \leq 2 \\
 & g_1(x) = x_0 + 3x_1 \leq 5 \\
 & g_2(x) = x_0^2 + x_1^2 - 2x_1 \leq 1 \\
 & x_i \geq 0, \quad i = 0, 1 \\
 & x_0 \in \mathbf{Z}
 \end{aligned} \tag{3.5}$$

该例子³仅要求 x_0 为整数。此时，需要通过 `set_enable_integer_filter` 方法来进行必要的配置：

```

1  #include <anyprog/anyprog.hpp>
2
3  int main(int argc, char** argv)
4  {
5      size_t dim = 2;
6      anyprog::optimization::function_t obj = [&](const anyprog::real_block& x) {
7          return 0.5 * x(0) * x(0) + 0.5 * x(1) * x(1) - 2 * x(0) - 2 * x(1);
8      };

```

³<https://inverseproblem.co.nz/OPTI/index.php/Probs/MIQCQP>

```

9
10 std::vector<anyprog::optimization::inequation_condition_function_t> ineq = {
11     [](const anyprog::real_block& x) {
12         return -x(0) + x(1) - 2;
13     },
14     [](const anyprog::real_block& x) {
15         return x(0) + 3 * x(1) - 5;
16     },
17     [](const anyprog::real_block& x) {
18         return x(0) * x(0) + x(1) * x(1) - 2 * x(1) - 1;
19     }
20 };
21
22
23 anyprog::optimization::range_t range = { 0, 100 };
24
25 anyprog::optimization opt(obj, range, dim);
26 opt.set_inequation_condition(ineq);
27 opt.set_filter_function([](anyprog::real_block& x) {
28     x(0) = round(x(0));
29 });
30 auto ret = opt.search(10,3);
31 anyprog::optimization::print(opt.is_ok(), ret, obj);
32 return 0;
33 }

```

其全局解为: $f(1, 1.33333) = -3.27778$ 。

3.5 TSP 问题

3.6 Assignment 问题

3.7 数据拟合问题

3.8 方程组求解

Chapter 4

展望