

AnyProg  
C++ 数学优化包

`pangpang@hi-nginx.com`

2019 年 12 月 22 日



# 目录

<b>1</b>	<b>起源</b>	<b>5</b>
<b>2</b>	<b>功能</b>	<b>7</b>
<b>3</b>	<b>技巧</b>	<b>9</b>
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 . . . . .	15
3.6	Assignment 问题 . . . . .	15
3.7	数据拟合问题 . . . . .	15
3.7.1	多项式拟合 . . . . .	15
3.7.2	类多项式拟合 . . . . .	16
3.7.3	非多项式拟合 . . . . .	18
3.8	方程组求解 . . . . .	21
<b>4</b>	<b>展望</b>	<b>23</b>



# 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：AnyProg<sup>1</sup>。

---

<sup>1</sup>Any 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**<sup>1</sup>、**QP**<sup>2</sup>、**NLP**<sup>3</sup>、**MILP**<sup>4</sup>、**MIQP**<sup>5</sup>、**MINLP**<sup>6</sup>。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。

---

<sup>1</sup>线性优化

<sup>2</sup>二次优化

<sup>3</sup>非线性优化

<sup>4</sup>混合整数线性优化

<sup>5</sup>混合整数二次优化

<sup>6</sup>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<sup>1</sup>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  {

```

<sup>1</sup>`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，它们并无差异——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。假设要求解 Rosenbrock<sup>2</sup>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  $\forall i, x_i \in [-10, 10]$ 。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     bool ok = false;
12     auto ret = anyprog::optimization::fminbnd(obj, range, dim, ok, 1e-10);
13     anyprog::optimization::print(ok, ret, obj);
14
15     return 0;
16 }

```

当然，若有较理想初值，也可使用 `fminunc` 求解：

```

1  #include <anyprog/anyprog.hpp>

```

<sup>2</sup>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>。

```

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    bool ok = false;
13    auto ret = anyprog::optimization::fminunc(obj, param, ok, 1e-10);
14    anyprog::optimization::print(ok, 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:  $\forall i, x_i \in \mathbf{Z}$ , 也包括混合整数优化:  $\exists i, x_i \in \mathbf{Z}$ 。所谓整数, 既指一般整数的情况:  $\exists i, x_i \in \mathbf{Z}$ , 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<sup>3</sup>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     };
9

```

---

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

```

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:  $f(1, 1.33333) = -3.27778$ 。

再看例子 (3.6):

$$\begin{aligned}
 \min_{x \in \mathbf{R}^2} \quad & f(x) = -x_1 + 2x_0 - \ln \frac{x_0}{2} \\
 \text{s.t.} \quad & g_0(x) = -x_0 - \ln \frac{x_0}{2} + x_1 \leq 0 \\
 & x_0 \in [0.5, 1.4], x_1 \in \{0, 1\}
 \end{aligned} \tag{3.6}$$


---

```

1  #include <anyprog/anyprog.hpp>
2
3  int main(int argc, char** argv)
4  {
5      anyprog::optimization::function_t obj = [&](const anyprog::real_block& x) {
6          return -x(1) + 2 * x(0) - log(x(0) / 2.0);
7      };
8
9      std::vector<anyprog::optimization::inequation_condition_function_t> ineq = {
10          [](const anyprog::real_block& x) {
11              return -x(0) - log(x(0) / 2.0) + x(1);
12          }
13      };
14
15
16      std::vector<anyprog::optimization::range_t> range = { { 0.5, 1.4 }, { 0, 1 } };
17
18      anyprog::optimization opt(obj, range);
19      opt.set_inequation_condition(ineq);

```

```

20     opt.set_filter_function([](anyprog::real_block& x) {
21         x(1) = round(x(1));
22     });
23     auto ret = opt.search(5,3);
24     anyprog::optimization::print(opt.is_ok(), ret, obj);
25     return 0;
26 }

```

其解为:  $f(1.37482, 1) = 2.12447$ 。

`set_enable_integer_filter` 有一个重载: 通过 `std::vector<size_t>` 容器提供需要整数化的变量分量索引列表参数, 即可完成整数优化配置。此法不仅更胜一筹, 而且对 `set_enable_binary_filter` 方法同样有效。

## 3.5 TSP 问题

## 3.6 Assignment 问题

## 3.7 数据拟合问题

数据拟合按照最小二乘法进行。我把一般拟合问题分为三类模型: 多项式拟合, 类多项式拟合和非多项式拟合。

### 3.7.1 多项式拟合

其数学模型为

$$y = \sum_{i=0}^m p_i x^i$$

调用 `anyprog::fit::solve` 方法即可轻松求解, 解为降幂系数形式。比如给定变量数据为 [3, 5, 7, 9, 11, 13], 待拟合数据为 [1.85, 2.1, 2.4, 2.5, 2.7, 2.8], 以二次多项式拟合之。那么,

```

1  #include <anyprog/anyprog.hpp>
2  #include <iostream>
3
4  int main(int argc, char** argv)
5  {
6      anyprog::real_block xdat(6, 1), ydat(6, 1);
7      xdat << 3, 5, 7, 9, 11, 13;
8      ydat << 1.85, 2.1, 2.4, 2.5, 2.7, 2.8;
9
10     anyprog::fit fit(xdat, 2);
11     auto ret = fit.solve(ydat), fitting = fit.fitting(ret);
12     std::cout << ret << "\n";
13     std::cout << "xdat\tydat\tfit\t\n";
14     for (size_t i = 0; i < xdat.rows(); ++i) {
15         std::cout << xdat(i) << "\t" << ydat(i) << "\t" << fitting(i) << "\t\n";
16     }
17 }

```

```

18     std::cout <<fit.r_squared(ydat,fitting) <<"\n";
19     return 0;
20 }

```

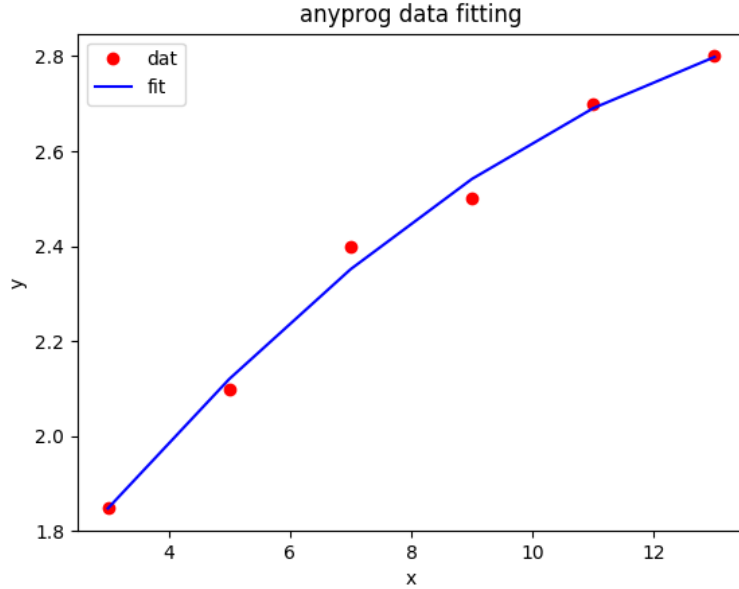


图 3.1: 多项式拟合

其解为:  $y = -0.00513393x^2 + 0.177143x + 1.36299$ , 决定系数为 0.992962, 待拟合数据和拟合数据比较如上图。

### 3.7.2 类多项式拟合

其数学模型为

$$y = \sum_{i=0}^m p_i f_i(x_0, \dots, x_n)$$

其中,  $f_i$  的函数形式不限于多项式。比如给定变量数据为 [3.6, 7.7, 9.3, 4.1, 8.6, 2.8, 1.3, 7.9, 10.0, 5.4], 待拟合数据为 [16.5, 150.6, 263.1, 24.7, 208.5, 9.9, 2.7, 163.9, 325.0, 54.3], 以函数  $y = p_0x^2 + p_1 \sin(x) + p_2x^3$  拟合之:

```

1  #include <anyprog/anyprog.hpp>
2  #include <fstream>
3  #include <iostream>
4
5  int main(int argc, char** argv)
6  {
7      anyprog::real_block xdat(10, 1), ydat(10, 1);
8      xdat << 3.6, 7.7, 9.3, 4.1, 8.6, 2.8, 1.3, 7.9, 10.0, 5.4;
9      ydat << 16.5, 150.6, 263.1, 24.7, 208.5, 9.9, 2.7, 163.9, 325.0, 54.3;
10     std::vector<anyprog::fit::function_t> fun = {
11         [](const anyprog::real_block& x, const anyprog::real_block& p) {
12             return pow(x(0, 0), 2);
13         },

```



```

14     [](const anyprog::real_block& x, const anyprog::real_block& p) {
15         return sin(x(0, 0));
16     },
17     [](const anyprog::real_block& x, const anyprog::real_block& p) {
18         return pow(x(0, 0), 3);
19     }
20 };
21 anyprog::real_block param(3, 1);
22 param.fill(0);
23 anyprog::fit fit(xdat, fun, param);
24 auto ret = fit.solve(ydat), fitting = fit.fitting(ret);
25 std::cout << ret << "\n";
26 std::cout << "xdat\tydat\tfit\t\n";
27 for (size_t i = 0; i < xdat.rows(); ++i) {
28     std::cout << xdat(i) << "\t" << ydat(i) << "\t" << fitting(i) << "\t\n";
29 }
30 std::cout << fit.r_squared(ydat, fitting) << "\n";
31 return 0;
32 }

```

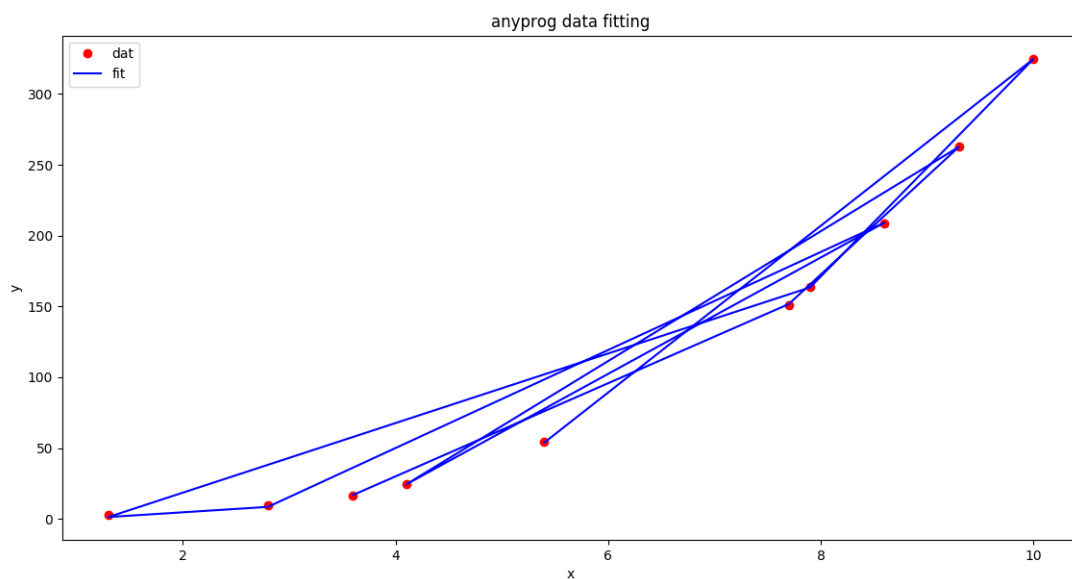


图 3.2: 类多项式拟合

其解为  $y = 0.2269x^2 + 0.338518 \sin(x) + 0.302151x^3$ ，决定系数为 0.999949。待拟合数据和拟合数据比较如上图。

### 3.7.3 非多项式拟合

其数学模型为

$$y = \sum_{i=0}^m p_i f_i(x_0, \dots, x_n, p_0, \dots, p_m)$$

其中,  $f_i$  包含参数  $p_j (j = 0, \dots, m)$  变量。拟合此类模型需要调用 `lssolve` 或者 `lssearch` 方法。

比如给定数据为:

```
( 0.683333333, 1.552133333 ),
( 1.066666667, 1.610833333 ),
( 2.166666667, 1.702533333 ),
( 3.333333333, 1.757483333 ),
( 6.366666667, 1.825533333 ),
( 12.466666667, 1.890833333 ),
( 43.88333333, 1.99507451 ),
( 58.53333333, 2.016733333 ),
( 118.7, 2.066433333 ),
( 138.75, 2.077583333 ),
( 479.6166667, 2.160233333 ),
( 499.6666667, 2.163333333 )
```

第一分量为变量数据, 第二分量为待拟合数据, 以函数  $y = \frac{p_0 \log(x)}{1 + p_1 \log(x)} + p_2$  拟合之:

---

```
1  #include <anyprog/anyprog.hpp>
2  #include <fstream>
3  #include <iostream>
4
5  int main(int argc, char** argv)
6  {
7      anyprog::real_block dat(12, 2);
8      dat << 0.683333333, 1.552133333,
9            1.066666667, 1.610833333,
10           2.166666667, 1.702533333,
11           3.333333333, 1.757483333,
12           6.366666667, 1.825533333,
13           12.466666667, 1.890833333,
14           43.88333333, 1.99507451,
15           58.53333333, 2.016733333,
16           118.7, 2.066433333,
17           138.75, 2.077583333,
18           479.6166667, 2.160233333,
19           499.6666667, 2.163333333;
20
21      std::vector<anyprog::fit::function_t> fun = {
22          [](const anyprog::real_block& x, const anyprog::real_block& p) {
23              return (p(0, 0) * log(x(0, 0)) / (1 + p(1, 0) * log(x(0, 0))) + p(2, 0)) / p(0, 0);
24          }
25      };
26      anyprog::real_block param(3, 1);
```

```

27 param.fill(0.1);
28 anyprog::fit fit(dat.col(0), fun, param);
29 auto ret = fit.lssolve(dat.col(1)), fitting = fit.fitting(ret);
30 std::cout << ret << "\n";
31 std::cout << "xdat\tydat\tfit\t\n";
32 for (size_t i = 0; i < dat.rows(); ++i) {
33     std::cout << dat(i, 0) << "\t" << dat(i, 1) << "\t" << fitting(i) << "\t\n";
34 }
35
36 std::cout << fit.r_squared(dat.col(1), fitting) << "\n";
37 return 0;
38 }

```

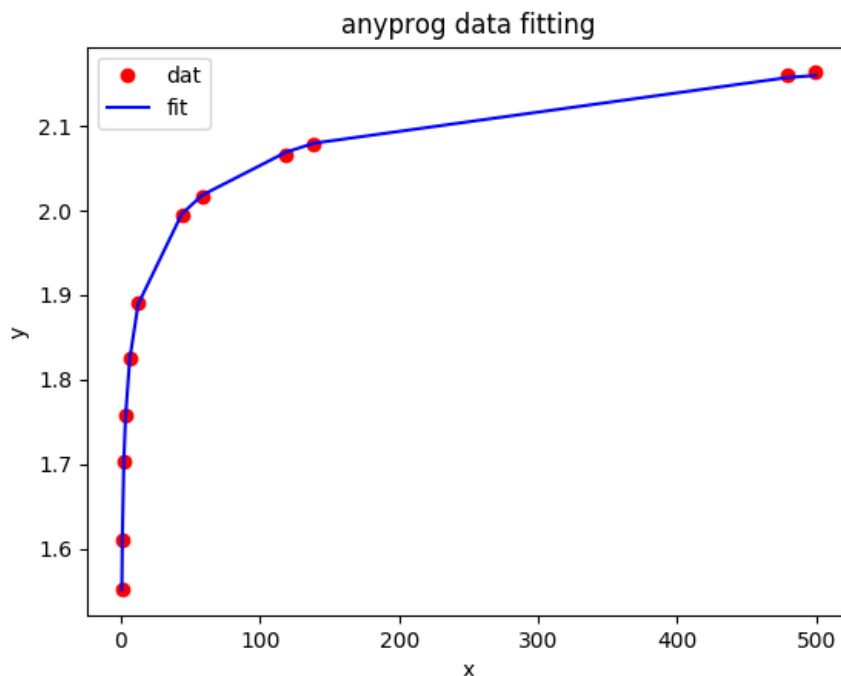


图 3.3: 非多项式拟合

其解为  $y = \frac{0.137182 \log(x)}{1 + 0.0864034 \log(x)} + 1.60523$ , 决定系数为 0.999875。待拟合数据和拟合数据比较如上图。因模型规范会自动在  $f_i$  前添加参数  $p_i$ , 故而需对待拟合模型作适当调整以矫正模型, 如第 23 行代码所示。

应对非多项式拟合的更为通用的方法是 `lssearch`, 它能进行全局优化拟合, 其解不依赖于初始解, 设置解搜索范围即可。其参数与全局优化方法 `search` 颇为相似。若 `lssolve` 所求局部解不理想, 可调用该方法。比如:

```

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

```

```

6 {
7   anyprog::real_block xdat = anyprog::block::linspace(0, 4, 100);
8   anyprog::real_block ydat = xdat.unaryExpr([](double x) {
9       return 1.5 * sin(x) + cos(M_PI * x);
10  });
11
12  std::vector<anyprog::fit::function_t> fun = {
13      [](const anyprog::real_block& x, const anyprog::real_block& p) {
14          return (p(0) * sin(x(0)) + cos(p(1) * x(0))) / p(0);
15      }
16  };
17
18  anyprog::real_block param(2, 1);
19  param << 1.0, 4.2;
20
21  std::vector<anyprog::optimization::range_t> range = { { 0, 10 }, { 0, 10 } };
22
23  anyprog::fit fit(xdat, fun, param);
24  auto ret = fit.lssearch(ydat, range, 10, 3), fitting = fit.fitting(ret);
25  std::cout << ret << "\n"
26      << fit.r_squared(ydat, fitting) << "\n";
27
28  return 0;
29 }

```

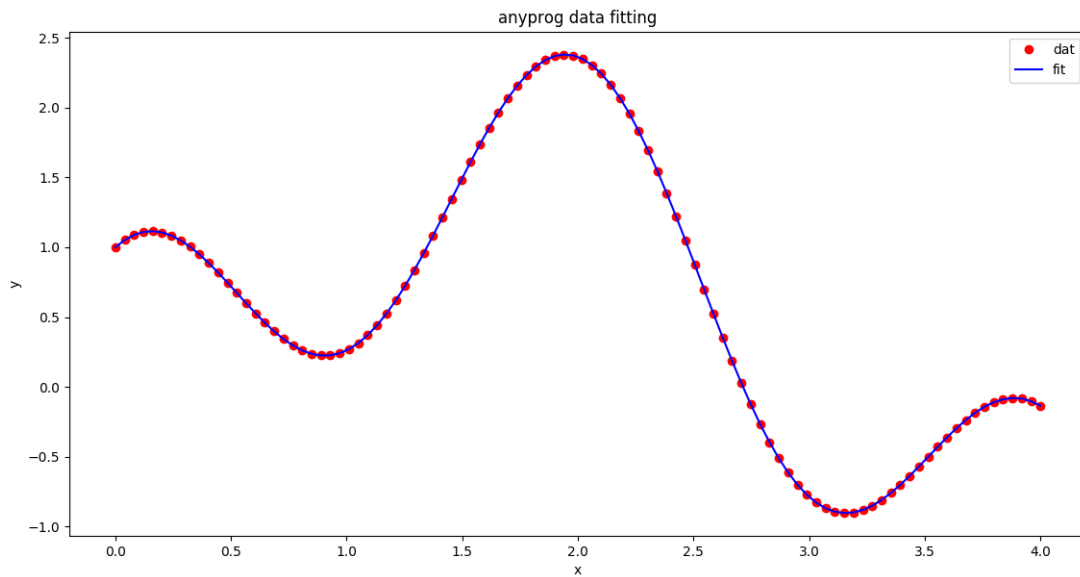


图 3.4: 非多项式全局优化拟合

此例先通过函数  $y = 1.5 \sin(x) + \cos(\pi x)$  在区间  $x \in [0, 4]$  生成 100 个数据，然后通过 `lssearch` 方法在空间

$\forall i, p_i \in [0, 10]^2$  进行全局搜索求解。编译运行该程序，可得决定系数为 1，问题获得完美解决。待拟合数据和拟合数据比较如上图。

## 3.8 方程组求解



## Chapter 4

### 展望