

订阅DeepL Pro以编辑此演示文稿。  
访问www.DeepL.com/Pro，了解更多信息。

CS100家庭作业8

# 编程入门，2021年春季

提交截止日期: 2021-06-22 23:59:59

1. poly.hpp**的实现**

2021-06-07

你要实现一个名为Poly的类，它简单地表示一个单变量、标量值的多项式。

多项式可以用一个系数向量来表示，以一个渐进的方式表示。举例来说。

x3+2x2+2可以用2*,* 0*,* 2*,* 1表示，因为*x* 0, *x* 1, *x* 2, *x* 3的系数分别是2*,* 0*,* 2*,* 1。我们给了你Poly类的接口，你需要在这里实现基本的函数和运算符。我们已经为它们提供了注释，而且它们也是很好的自我解释。

*{}*

在Poly类中还有一些值得关注的功能。

* 1. 函数评估这个多项式的值，作为参数的一个函数。
  2. 函数der构造了多项式的一阶导数。
  3. 最重要的是

1 std :: pair < T,T> operator ()( T param ) const.

返回param处的多项式的值和param处的一阶导数（只是对导数多项式的评估）。这个函数对问题2很有用。

**注意**：与函数式编程有一个错综复杂的联系：多项式可以被解释为一个简单的函数，它接收一个参数并返回一个标量。 更有趣的是，两个标量多项式的和、积等甚至组合仍然是一个标量多项式，因此对这些多项式的许多操作都可以理解为函数形式。 然而，并不是所有的函数形式都可以轻易实现，因为两个多项式的除法并不返回多项式。

# **提交**：请向autolab提交一个包含poly.hpp的.tar文件。

1. functional\_forms.hpp**的实现**

在这里，我们将多项式视为函数形式，并使用λ函数，即一个或两个单变量函数，并从中构造一个新的单变量函数。需要定义六个基本操作。

* 总数
* 减法
* 乘法
* 划分
* 功能构成
* 电源功能

这些函数应该很容易理解：如果*f*（*x*）=x2+1，*g*（*x*）=*x 3*，那么（*f*+*g*）（*x*）=x2+*x 2*，（*f*（*g*））（*x）*=x2 *6x*+10。*(验证一下吧！)*

*−*

*−−*

所涉及的函数有一个特殊性。对于所有这些函数，我们假设它们返回一个std::pair<double,double> ，其中第一个代表该函数在某一点上的实际值，第二个是一阶导数。

例如：如果*f* (*x)* = x2 + 1，*g*(*x)* = *x -* 3，那么调用derSum(f, g)(0.5)将返回一个std::对

*(*−1*.*25*,* 2).*(验证一下吧！)*

**注意事项**。

* 1. 对于所有上述函数形式，导数是明确定义的。

如果*g*（*x）*=*f*（*x）*+*h*（*x*），那么*dg*（*x）*=*df*（*x）*+*dh*（*x）*。

*dxdxdx*

如果*g*（*x）*=*f*（*x*）*-h*（*x*），那么*dg*（*x）*=*df*（*x*）*-dh*（*x）*。

*dx*

*dx*

*dx*

如果*g*(*x)*=*f*(*x)-h*(*x)*，那么*dg*(*x)*=*df*(*x)h*(*x)*+*f*(*x)dh*(*x*)

*dx*

*dx*

*dx*

如果*g*(*x)*=

*f* (*x*)

*h*(*x*)

，那么

*dgdf* (*x*)*h*(*x*) *f* (*x*) *dh* (*x*)

(*x*)=  *dxdx*

*−*

*dxh*(*x*) *- h*(*x*)

如果*g*（*x）*=*h*（*f*（*x）*），那么*dg*（*x）*=*dh*（*f*（*x）*）*df*（*x*）。

*dxdxdx*

如果*g*（*x）*=（*f*（*x*））*p*，那么*dg*（*x*）=*p*（*f*（*x）*）（*p*-1）*df*（*x*）。

*dxdx*

* 1. 多项式实现了operator()，它返回std::pair<double,double>（如果double是模板参数）。它们被用来作为上述函数形式的输入。

使用上述函数形式的好处是，我们有了自动微分的功能!在文件test lambdas.cpp中有一个例子，它使用了下面这个复杂的函数。

(p2(p1(*x)*))3 *-* p3(*x*)

*g*(*x)* =

P4(*X)* + P5(*X*) *-* P6(*X*)

由于表达式中含有除法，所以不可能将多项式组合成新的多项式。此外，手工推导导数会很繁琐，而且用有限差分计算导数是不准确的。然而，通过使用上述λ函数，计算任何给定点的值或导数都将变得简单。

**提示**：（这不是你实现的一部分）如果你想验证你的微分的正确性，你可以通过使用有限差分近似法轻松实现。

*dg*

＝四分之一

*g*(*x* + ∆*x*) *- g*(*x*)

*dx* ∆*x→*0 ∆*x*

# **提交：**请向autolab提交一个包含 "poly.hpp "和 "functional\_forms.hpp "的.tar文件。

1. Newton.hpp**的实现**

这里你需要实现牛顿方法（对于简单的单变量函数）。这是一种通过迭代使用线性逼近的方法从一个起点找到最近的根。你可以按照下面的链接，或者在任何搜索引擎上搜索 "牛顿方法"[。https://en.wikipedia.org/wiki/Newton%27s\_method](https://en.wikipedia.org/wiki/Newton%27s_method)

其基本思想是，我们从

，并通过反复寻找下一个*x*

直到*f*（*xn*）足够接近于0。

x0 = startingPoint

*xk*+1 =*xk -* ( *f*（*xk*）*/ft*（*xk）*)

我们可以很容易地将其应用于问题2中的函数，因为它们返回值和导数。这个函数的第一个参数（f）正是问题2中生成的lambda函数。你可以参考所提供的例子test\_newton.cpp中的用法，了解详细情况。

# **提交**：请向autolab提交一个包含 "poly.hpp"、"functional\_forms.hpp "和 "newton.hpp "的.tar文件。

图1：牛顿的方法