03_28_numer_interpol_diff_int

Unknown Author

April 1, 2014

Part I

Numeric Interpolation, Differentiation and Integratio

1 Interpolation

Suppose you know that a function takes the values given in the following table

x	y=f(x)
0	0
1	1
2	8
3	27
4	64
5	125

Problem : Find a polynomial p such that $f(x) = p(x)$ for the given values of x.

There are infinitely many such polynomials. One such polynomial can be constructed using Langrange's interpolation formula. Suppose we are given n points $x_1, \ldots, x_n$ and the value of f at these points $y_1, \ldots, y_n$.

Define

$$l_i(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)} \tag{1}$$

These are definitely polynomials which take the value 1 at x_i and 0 at all other x_j's. We define

$p(x) = \sum_{i=1}^{n} y_i l_i(x)$ to be the required polynomail.

This is called the Lagrange interpolation formula.

In [1]:
```
# We create a class called interpolation.

class LagrangeInterpolation :
    """Creates a class for lagrange interpolation. Creates a callable polynomial.

    Attributes :
        l : a list of (x, f(x)) pairs.
```

```
    Special methods define :
        __init__
        __call__
        __str__

    Other methods :
        add_pair(p) : Add pair to the list l
        poly() : Returns a list containing the coefficients of the polynomial.
    """

    def __init__(self, l) :
        self.l = l

    def add_pair(self, p) :
        self.l.append(p)

    def poly_add(self, l1, l2) :
        """Given two polynomial coeff lists, it adds them."""
        len1 = len(l1)
        len2 = len(l2)
        max1 = max(len1, len2)
        rlist = []
        for i in range(max1) :
            if i < len1 and i < len2 :
                rlist.append(l1[i] + l2[i])
            elif i < len1 :
                rlist.append(l1[i])
            elif i < len2 :
                rlist.append(l2[i])
            else :
                print "Serious error. Execution should not reach here."
                rlist = None
        return rlist

    def poly_mult(self, l1, l2) :
        """Product of two poly coeff lists."""
        len1 = len(l1)
        len2 = len(l2)
        prodlen = len1 + len2 - 1 # Why?
        #print len1, len2, prodlen

        prod= []
        for i in range(prodlen) :
            i_th_coeff = 0
            for j in range(i+1) :
                #print i, j
                if j < len1 and i - j < len2 and j >= 0 and i >= j:
                    #print j, i-j, len1, len2
                    i_th_coeff += l1[j] * l2[i - j]
            prod.append(i_th_coeff)
        return prod

    def li(self, i) :
        l = self.l
        lenlist = len(l)
        if i >= lenlist :
            print "i = %d is out of range." % i
            li = None
        else :
            li = [1.0]
            xi = float(l[i][0])
            for j in range(lenlist) :
                if j != i :
                    xj = float(l[j][0])
                    #print i, j, xi, xj, li
```

```
                    li = self.poly_mult(li, [-(xj/(xi - xj)), (1.0/(xi - xj))])
        return li

    def poly(self) :
        l = self.l
        lenl = len(l)
        p = [0.0]
        for i in range(lenl) :
            yi = [float(l[i][1])]
            ith_term = self.poly_mult(yi, self.li(i))
            p = self.poly_add(p, ith_term)
            #print ith_term, p
        return p

    def __call__(self, t) :
        l = self.l
        ll = len(l)
        valatx = 0
        x = []
        y = []
        for k in range(ll) :
            x.append(float(l[k][0]))
            y.append(float(l[k][1]))
        for i in range(ll) :
            ithterm = y[i]
            for j in range(ll) :
                if j != i :
                    ithterm *= (t - x[j]) / (x[i] - x[j])
            valatx += ithterm
        return valatx

    def __str__(self) :
        pl = self.poly()
        lpl = len(pl)
        strval = ""
        for i in range(lpl) :
            if i == 0 :
                strval += "%g" % pl[i]
            else :
                strval += " + %g x^%d" % (pl[i], i)
        return strval

    def __repr__(self) :
        return "LangrangeInterpolation(" + str(self.l) + ")"
```

In [2]:

```
if __name__ == '__main__' :
    cl = LagrangeInterpolation([(1, 0), (2, 15), (3, 80)])
    print "Testing Interpolation :"
    print "Polynomial : "
    print cl.poly()
    cl.add_pair((-1, 0))
    cl.add_pair((10, 9999))
    print "_____________________________"
    print "Polynomial after adding two points."
    print cl.poly()
    print "Value of the polynomial at .01 :", cl(.01)
    print "Testing __str__ :",
    print cl
    print "Testing __repr__ :",
    print cl.__repr__()
```

```
Testing Interpolation :
Polynomial :
[35.0, -60.0, 25.0]
_____________________________
```

```
Polynomial after adding two points.
[-0.9999999999999982, -5.329070518200751e-15, -3.552713678800501e-15,
3.552713678800501e-15, 1.0000000000000002]
Value of the polynomial at .01 : -0.99999999
Testing __str__ : -1 + -5.32907e-15 x^1 + -3.55271e-15 x^2 +
3.55271e-15 x^3 + 1 x^4
Testing __repr__ : LangrangeInterpolation([(1, 0), (2, 15), (3, 80),
(-1, 0), (10, 9999)])
```

2 Differentiation

We shall be differentiating with help of the approximation

$$f'(x) \simeq \frac{f(x+h) - f(x-h)}{2h} \tag{2}$$

We implement that too as a class.

In [3]:
```
class Derivative2 :
    """New class for derivatives

    Attributes : f, h, err, max_iter, show_steps
    Methods : seth, seterr, setiter, switch_show_steps
    """
    def __init__ (self, f) :
        self.f = f
        self.h = 1
        self.err = 1E-5
        self.max_iter = 10000
        self.show_steps = False

    def seth(self, h) :
        self.h = h
        return None

    def seterr(self, e) :
        self.err = e
        return None

    def setiter(self, N) :
        self.max_iter = N
        return None

    def switch_show_steps(self) :
        if self.show_steps :
            self.show_steps = False
        else :
            self.show_steps = True
        return None

    def _basic_der(self, y, d) :
        f = self.f
        return (f(y + d) - f(y - d))/(2*d)

    def __call__(self, x) :
        d = self.h
        f = self.f
        e = self.err

        df = self._basic_der(x, d)
        cont_loop = True
```

```
        no_iter = 0
        while cont_loop :
            no_iter += 1
            d /= 2.0
            dfnew = self._basic_der(x, d)
            if abs(df - dfnew) < e :
                retval = dfnew
                cont_loop = False
            elif no_iter > self.max_iter :
                print "Exceeded the maximum number of iterations."
                retval = None
                cont_loop = False
            else :
                df = dfnew
                if self.show_steps :
                    print "%06d. df = %10g" % (no_iter, df)
        return retval

    def __str__(self) :
        return "Cannot display the deriviative."
```

In [4]:

```
if __name__ == '__main__' :
    der = Derivative2(lambda x : (1.0/3)*x*x*x + x)
    print "der(0) =", der(0), ", der(1) =", der(1)
    der.switch_show_steps()
    print "der(10) with steps:"
    print der(10)
    der.seterr(1E-50)
    der.seth(1E8)
    der.setiter(10)
    der.switch_show_steps()
    print "der(1.5) with err = 1E-50, h = 1E8, max_iter = 10 :", der(1.5)
    der.switch_show_steps()
    print "Same thing with steps :"
    print der(1.5)
```

```
der(0) = 1.00000127157 , der(1) = 2.00000127157
der(10) with steps:
000001. df =    101.083
000002. df =    101.021
000003. df =    101.005
000004. df =    101.001
000005. df =        101
000006. df =        101
000007. df =        101
000008. df =        101
101.000001272
der(1.5) with err = 1E-50, h = 1E8, max_iter = 10 : Exceeded the
maximum number of iterations.
None
Same thing with steps :
000001. df = 8.33333e+14
000002. df = 2.08333e+14
000003. df = 5.20833e+13
000004. df = 1.30208e+13
000005. df = 3.25521e+12
000006. df = 8.13802e+11
000007. df = 2.03451e+11
000008. df = 5.08626e+10
000009. df = 1.27157e+10
000010. df = 3.17891e+09
```

```
Exceeded the maximum number of iterations.
None
```

3 Integration

We shall use the *trapezoidal* method for integration. Suppose we want to integrate a function $f : \mathbb{R} \to \mathbb{R}$ from a to b. The method involves partitioning the interval $[a, b]$ into N equal intervals $[a, a+(b-a)/N]$, $[a+(b-a)/N, a+2(b-a)/N]$, $\ldots$, $[a+(N-1)(b-a)/N, b]$. Now the assumption is that when N is sufficiently large, the intervals are so small that f is practically linear. Thus to integrate f on the interval $[l_i = a+(i-1)(b-a)/N, r_i = a+i(b-a)/N]$, one just has to find the integral of the line joining $(l_i, f(l_i))$ and $(r_i, f(r_i))$ for $i = 1, 2, \ldots, N$. The integral of each such line is $0.5(r_i - l_i)(f(r_i) + f(l_i))$. Summing these over all i we get the integral.

Thus we get the equation

$$\int_a^b f dx = \sum_{i=1}^{N} \int_{a+(i-1)\frac{b-a}{N}}^{a+i\frac{b-a}{N}} f dx \simeq \sum_{i=1}^{N} \frac{b-a}{2N} \left(f(a+(i-1)\frac{b-a}{N}) + f(a+i\frac{b-a}{N}) \right) = \frac{b-a}{2N} \sum_{i=1}^{N} \left(f(a+(i-1)\frac{b-a}{N}) + f(a+ \right. \tag{3}$$

The right hand side can be rewritten as

$$S_N := \frac{b-a}{N} \frac{f(a)+f(b)}{2} + \frac{b-a}{N} \sum_{i=1}^{N-1} f(a+i\frac{b-a}{N}) \tag{4}$$

Saving computations.

Note that if we call the expression on the right hand side to be S_N, we have (check!) :

$$S_{2^{n+1}} = \frac{1}{2} S_{2^n} + \frac{b-a}{2^{n+1}} \sum_{i=0}^{2^n-1} f(a+(2i+1)\frac{b-a}{2^{n+1}}) = \frac{1}{2} S_{2^n} + \frac{b-a}{2^{n+1}} T_{n+1} \tag{5}$$

We shall vary n starting from n = 1 and stop when two consecutiive S_{2^n}'s are close enough.

In [5]:
```
class Integrate :
    """Given a function, and one end point it computes the integral to a point x"
    Attributes :
        f : function
        a : starting point of integration
        b : end point of integration
        max_iter : maximum number of iterations
        err : Error below which everything is considered zero.
        showstep : show steps

    Methods :
        integral(a, b) : Integral of f from a to b
        seta : sets a
        setb : Sets b
        setiter : sets max_iter
        seterr : sets error
        switch_showstep : Switches showstep
        __call__(x) : returns the integral from a preset a to x
    """

    def __init__(self, f, a) :
        self.f = f
```

```
        self.a = a
        self.b = 0
        self.max_iter = 10000
        self.err = 1E-5
        self.showstep = False

    def seta(self, val) :
        self.a = val
        return None

    def setb(self, val) :
        self.b = val
        return None

    def setiter(self, N) :
        self.max_iter = N
        return None

    def seterr(self, e) :
        self.err = e
        return None

    def switch_showstep(self) :
        if self.showstep :
            self.showstep = False
        else :
            self.showstep = True

    def integral(self, p, q) :
        f = self.f
        n = 1
        Sn = (q-p)*(f(p) + f(q))/4.0 + (q-p)*f((p+q)/2.0)/2.0
        halfpowern = 0.5
        twopowernm1 = 1
        cont_loop = True
        while cont_loop :
            n += 1
            halfpowern /= 2.0
            twopowernm1 *= 2
            Snm1 = Sn # S_{n-1}
            Tn = 0
            for k in range(twopowernm1) :
                Tn += f(p + (2*k + 1) * (q - p) * halfpowern)
            Sn = Snm1/2.0 + halfpowern * Tn * (q-p)

            if abs(Sn - Snm1) < self.err :
                retval = Sn
                cont_loop = False
            elif n >= self.max_iter :
                print "Maximum number of iterations exceeded."
                retval = None
                cont_loop = False
            else  :
                if self.showstep :
                    print "%07d. N = %20d, S_N = %25g." % (n, twopowernm1 * 2, Sn)
        return retval

    def __call__(self, x) :
        self.setb(x)
        return self.integral(self.a, self.b)

    def __str__(self) :
        return "Cannot display the integral."
```

In [6]:

```
if __name__ == '__main__' :
    i = Integrate(lambda x : x*x, 0)
    i.switch_showstep()
    i.seterr(1E-10)
    i.setiter(10)
    print "Integral of x^2 between 0 and 100 with steps :"
    print i.integral(0, 100)
    i.setiter(10000)
    i.seterr(1E-5)
    print "i(3) ="
    print i(3)
    print "i =", i
    i.switch_showstep()
    print "i(6) =", i(6)
```

```
Integral of x^2 between 0 and 100 with steps :
0000002. N =                    4, S_N =                    343750.
0000003. N =                    8, S_N =                    335938.
0000004. N =                   16, S_N =                    333984.
0000005. N =                   32, S_N =                    333496.
0000006. N =                   64, S_N =                    333374.
0000007. N =                  128, S_N =                    333344.
0000008. N =                  256, S_N =                    333336.
0000009. N =                  512, S_N =                    333334.
Maximum number of iterations exceeded.
None
i(3) =
0000002. N =                    4, S_N =                   9.28125.
0000003. N =                    8, S_N =                   9.07031.
0000004. N =                   16, S_N =                   9.01758.
0000005. N =                   32, S_N =                   9.00439.
0000006. N =                   64, S_N =                    9.0011.
0000007. N =                  128, S_N =                   9.00027.
0000008. N =                  256, S_N =                   9.00007.
0000009. N =                  512, S_N =                   9.00002.
0000010. N =                 1024, S_N =                        9.
9.00000107288
i = Cannot display the integral.
i(6) = 72.0000021458
```