Yanes Garcia
Mathematics 4670

Divided-difference method

1. In this problem, we are using the divided-difference method, which successively generates the polynomials themselves. Suppose we are given the n+1 points (x0, f(x0)), (x1, f(x1)), …, (xn, f(xn)). There are n+1 zeroth divided differences of the function f. For each i= 0, 1, …, n we define f[xi] simply as the value of f at xi:

f[xi] = f(xi)

The remaining divided differences are defined inductively. There are n first divided differences of f, one for each i= 0, 1, …, n-1. The first divided difference relative to xi and xi+1 is denoted f[xi, xi+1] and is defined by

f[xi, xi+1]= (f[xi+1]-f[xi])/(xi+1 – xi)

After the (k-1)st divided differences,

f[xi, xi+1, xi+2, …, xi+k-1] and f[xi+1, xi+2, …, xi+k-1, xi+k]

have been determined, the kth divided difference relative to xi, xi+1, xi+2, …, xi+k is defined by

f[xi, xi+1, …, xi+k-1, xi+k]= (f[xi+1, xi+2, …, xi+k]- f[xi, xi+1, …, xi+k-1])/(xi+k –xi)

The process end with the single nth divided difference,

F[x0,x1,…,xn]=(f[x1,x2,…,xn]-f[x0,x1,…,xn-1])/(xn-x0)

With this notation, it can be shown that the nth Langrange interpolation polynomial for f with respect to x0,x1,…,xn can be expressed as

Pn(x)=f[x0] + f[x0,x1](x-x0)+f[x0,x1,x2](x-x0)(x-x1)+…+f[x0,x1,…,xn](x-x0)(x-x1)…(x-xn-1)

It is called Newton's divided-difference formula.

In the handout describing the simple method for finding interpolating polynomials we had a subroutine named createa with header:

subroutine createa (n, xdata, ydata, a)

This subroutine calculates the coefficients of the polynomial. In my case, I am going to do a similar thing but using the Newton divided difference method. My subroutine is named divdiff, and its header is:

subroutine divdiff (n, xdata, ydata, a)

This subroutine finds the coefficients using the recursive difference idea. Inside the subroutine, we create the two index table:

```
double precision, allocatable, dimension(:,:) :: T
```

Also, just before we return from the subroutine, we have to save our results into the one dimensional array a (indexed zero to n):

```
do i=0,n
a(i)=T(i,i)
end do
```

Finally, we have to deallocate the two dimensional array:

```
deallocate(T)
```

My fortran subroutine is:

```
subroutine divdiff (n, xdata, ydata, a)
implicit none
integer :: n,i,j
double precision :: xdata(0:n), ydata(0:n), a(0:n)
double precision, allocatable, dimension(:,:) :: T

allocate( T(0:n,0:n))

do i=0, n
T(i,0)= ydata(i)
end do

do j=1,n
 do i=j,n
T(i,j)=(T(i,j-1)-T(i-1,j-1))/(xdata(i)-xdata(i-j))
end do
end do

do i=0,n
a(i)=T(i,i)
end do

deallocate(T)
return
end
```

It needs to be tested. Let the data points be (3,1), (5,7), (6,2) a hand calculation of the type we have done in class reveals that a0=1, a1=3, a2=-2.666. We can check the subroutine with the following main program:

```
program testdivdiff
implicit none
integer :: n,i
```

```
double precision, allocatable, dimension(:) :: xdata, ydata, a
n=2
allocate( xdata(0:n), ydata(0:n), a(0:n))

xdata= (/ 3.0d0, 5.0d0, 6.0d0 /)
ydata= (/ 1.0d0, 7.0d0, 2.0d0 /)

call divdiff(n, xdata, ydata, a)

print*,"Coefficients from divdiff"

do i= 0, n
  print*, a(i)
end do

deallocate(xdata, ydata, a )
end program testdivdiff
```

Compiling and running this code produces the following output

```
Coefficients from divdiff
     1.00000000000
     3.00000000000
    -2.66666666667
```

2. Using the subroutine from problem one: subroutine divdiff (n, xdata, ydata, a). We can reproduce the table 3.9 from the text book.

subroutine divdiff (n, xdata, ydata, a)

My subroutine is:

```
subroutine divdiff(n, xdata, ydata, a)
implicit none
integer :: n,i,j
double precision :: xdata(0:n), ydata(0:n), a(0:n)
double precision, allocatable, dimension(:,:) :: T

allocate( T(0:n,0:n))

do i=0, n
T(i,0)= ydata(i)
end do
```

```
do j=1,n
  do i=j,n
T(i,j)=(T(i,j-1)-T(i-1,j-1))/(xdata(i)-xdata(i-j))
end do
end do

do i=0,n
a(i)=T(i,i)
end do

do i= 0, n
write(*,*) i, xdata(i), ydata(i), (T(i,j), j=1,n)
print*," "
end do

deallocate(T)
return
end
```

My driver program is:

```
program testdivdiff
implicit none
integer :: n
double precision, allocatable, dimension(:) :: xdata, ydata, a

n=4
allocate( xdata(0:n), ydata(0:n), a(0:n))

xdata= (/ 1.0d0, 1.3d0, 1.6d0, 1.9d0, 2.2d0 /)
ydata= (/ 0.7651977d0, 0.6200860d0, 0.4554022d0, 0.2818186d0,
0.1103623d0 /)

call divdiff(n, xdata, ydata, a)

deallocate(xdata, ydata, a )
end program testdivdiff
```

The first divided difference involving x0 and x1 is: f [xo, x1] = (f[x1]-f[x0])/(x1-x0). The remaining first divided differences are found in similar manner and are shown in the fourth column in Table 3.9. The second divided difference involves x0, x1, and x2, and the remaining second divided differences are shown in the fifth column. The third divided difference involving x0,x1,x2, and x3, and the fourth divided difference involves all the data points.

Compiling and running this code produces the following output

```
 0          1.00000000000          0.765197700000          0.00000000000
 0.00000000000            0.00000000000           0.00000000000

 1          1.30000000000          0.620086000000          -0.483705666667
 0.00000000000            0.00000000000           0.00000000000

 2          1.60000000000          0.455402200000          -0.548946000000
 -0.108733888889          0.00000000000           0.00000000000

3          1.90000000000          0.281818600000          -0.578612000000
-4.944333333333E-02       6.587839506173E-02       0.00000000000

4          2.20000000000          0.110362300000          -0.571521000000
1.181833333333E-02       6.806851851852E-02       1.825102880660E-03
```

3. We are going to use my subroutine divdiff to find the coefficients of the polynomial of smallest degree that interpolates the following data.

x	y
1	-4
2	-11
3	-14
4	-7
5	16

Using the subroutine and the main program from problem one, but changing the values of the x and y, we get the next coefficients:

```
Coefficients from divdiff
          -4.00000000000
          -7.00000000000
           2.00000000000
```

Which is correct.

Now that we have a working routine to calculate the coefficients of the interpolating polynomial, we need a way to evaluate that polynomial for a given x value. A method called nested multiplication will be used.

a0 + a1(x-x0) + a2(x-x0)(x-x1) + a3(x-x0)(x-x1)(x-x2) = a0+ (x-x0)(a1+a2(x-x1) + a3(x-x1)(x-x2))= a0+ (x-x0)(a1+(x-x1)(a2+ a3(x-x2)))

We can set some variable p equal to a3 then set p equal to p times (x-x2) then add a2. Next set p equal to p times x-x1 and add a1. Then set p equal p times x-xo and add a0.

My fortran function is:

```
double precision function p(n, xdata, a, x)
implicit none
integer :: i,n
double precision :: x, xdata(0:n), a(0:n)

p= a(n)
do i= n, 1, -1
  p= p*( x- xdata(i-1) ) + a(i-1)
end do

return
end
```

To test this we have to make a modification of my main program. My program is:

```
program testdivdiff
implicit none
integer :: n,i
double precision, allocatable, dimension(:) :: xdata, ydata, a
double precision :: p,x

n=2
allocate( xdata(0:n), ydata(0:n), a(0:n))

xdata= (/ 1.0d0, 2.0d0, 3.0d0, 4.0d0, 5.0d0 /)
ydata= (/ -4.0d0, -11.0d0, -14.0d0, -7.0d0, 16.0d0 /)

call divdiff(n, xdata, ydata, a)

do i= 0, n
x = xdata(i)
print*, xdata(i), ydata(i), ydata(i)-p(n,xdata,a,x)
end do

deallocate(xdata, ydata, a )
end program testdivdiff
```

Testing function p. We will print xdata(i), ydata(i), and ydata(i)-p(n,xdata,a,x). If p is working correctly that last column should all be zeros up to rounding errors

```
  1.00000000000        -4.00000000000        0.00000000000
```

2.00000000000	-11.0000000000	0.00000000000
3.00000000000	-14.0000000000	0.00000000000

4. When we are calculating divided differences as we have indicated, and all the divided differences in column number k were zero, the divided differences in later columns are also going to be zero. There is a relation between the finite divided differences and the derivatives of the interpolating polynomials. The nth derivative of an nth order polynomial is constant and the (n+1) st derivative is zero. We are going to start from x0 to xn, and we are going to look for the divided difference from zeroth to kth columns.

Zeroth divided difference:

f[xi]= f (xi)

First divided difference:

F[xi,xi+1] =(f[xi+1] – f[xi])/(xi+1 – xi)

Second divided difference:

F[xi,xi+1, xi+2] =(f[xi+1, xi+2] – f[xi, xi+1])/(xi+2 – xi)

Kth divided difference:

F[xi,xi+1, …, xi+k] =(f[xi+1, xi+2,…, xi+k] – f[xi, xi+1, …, xi+k-1])/(xi+k – xi)

When we get that all the divided difference in columns number k were zero, the divided differences in later columns are also going to be zero. It happens because if we input i=0,1,…,n, we cannot expect divided differences other than 0 when k>n. If all the divided differences in column number k were zero, we can affirm that k>n. The later columns are going to be k+1, k+2,… what means that they are going to continue being bigger than n. In all those cases, we expect that all the divided differences are zero. For instance, if we are given the data set (xi,f(xi)), i=0,1,2,3,4,5, and we try to find the 5th order Newton Divided Difference we get f[x0,x1,x2,x3,x4,x5]. However, if we try to find the 6th order Newton Divided Difference, we get that all the divided differences in the 6th column are zero. The column number k is not inside of my set data. In this example, k>5.