Physics 40: Laboratory Three

Tuesday, April 7, 2020

Today's Goals: Functions; Codes for Integration and Differentiation; Molecular Dynamics 1.

Much of physics is about integrating differential equations, so we need to know how to do calculus on a computer. Today, we start with simple integrals and derivatives, and introduce 'molecular dynamics'. We will also learn about 'functions' in C.

We will begin by modifying a code you wrote last week.

VERY IMPORTANT POLICY:

Never do a direct modification of a code you already have working. **Always** make a copy of the code (that is a new 'version') and change that. The reason is that if you introduce a bug as you make the modifications, you want to be able to go back to your original working code.

- Walk through the exponential code.

```
#include <stdio.h>
#include <math.h>
int main(void)
{
     int j,N;
     long int fact;
     double x,sum;
     printf("Enter N \n");
     printf("\n");
     scanf("%i",&N);
     printf("Enter x\n");
     scanf("%lf",&x);
     sum=1.;
     fact=1;
     for (j=1; j<N; j=j+1)
     {
     fact=fact*j;
     sum=sum+pow(x,j)/fact;
     printf("\n  %i  %lf",j,sum);
     }
     printf("\n");
     return 0;
}
```

- Molecular Dynamics Background.

Notice that in this old code to get the exponential, we computed the factorials in the main program. We will write a new version of the exponential code which isolates the computation of factorials in a separate function.

[1] Begin by writing a function for factorials:

```
#include <stdio.h>
#include <math.h>
int factorial();

int main(void)
{
    int i,N,fact;
    printf("\n Enter N    ");
    scanf("%i",&N);
    fact=factorial(N);
    printf("\n  N!=  %16i \n",fact);
    return 0;
}

int factorial(int N)
{
int j,fact=1;
for (j=2;j<N+1;j=j+1)
{
     fact=fact*j;
}
return fact;
}
```

[2] Then use it in your exponential function:

```
#include <stdio.h>
#include <math.h>
int factorial();

int main(void)
{
     int j,N,fact;
     double x,sum;
     printf("Enter N \n");
     scanf("%i",&N);
     printf("Enter x\n");
     scanf("%lf",&x);
     sum=1.;
     for (j=1; j<N; j=j+1)
     {
         fact=factorial(j);
         sum=sum+pow(x,j)/fact;
     }
     printf(" exp(x)=  %lf  \n",sum);
     return 0;
}


int factorial(int N)
{
int j,fact=1;
for (j=2;j<N+1;j=j+1)
{
     fact=fact*j;
}
return fact;
}
```

This looks a bit more complicated than the original exponential function program, but there are three **very good** reasons for writing codes this way. The first is readability. The second code is more clear because it isolated the calculation of the factorial in its own separate place. The second is debuggability. Writing codes in this way one can more easily write and debug the pieces individually. The third is conciseness. One often uses things like the factorial at many places in a program. Using the function approach, one does not need to place the lines for computing the factorial at multiple locations in the code.

WARNING: The code in [2] is a bit confusing. When the code encounters the line:
fact=factorial(j);
it passes the argument "j" to the function
int factorial(int N)
When "j" arrives at the function its value is assigned to "N", the argument of the function. Thus "N" in the function has the value "j" had in main. That's good because we want to compute and use j factorial in the calculation of e^x.

The (loop) variable "j" in the function is completely unrelated to the variable "j" in main.

I could have written the code a different way to make it more clear that "N" in the function is the same as "j" in main, and that "j" in the function is unrelated to "j" in main. (See below.) But in a way, the original 'confusing' version teaches an important lesson about variables in different pieces of a code.

```
#include <stdio.h>
#include <math.h>
int factorial();

int main(void)
{
     int j,N,fact;
     double x,sum;
     printf("Enter N \n");
     scanf("%i",&N);
     printf("Enter x\n");
     scanf("%lf",&x);
     sum=1.;
     for (j=1; j<N; j=j+1)
     {
         fact=factorial(j);
         sum=sum+pow(x,j)/fact;
     }
     printf(" exp(x)=  %lf  \n",sum);
     return 0;
}

int factorial(int j)
{
int k,fact=1;
for (k=2;k<j+1;k=k+1)
{
     fact=fact*k;
}
return fact;
}
```

[3] A code for differentiation! This code implements the usual definition of the derivative.

$$\frac{df}{dx} \equiv \lim_{dx \to 0} \frac{f(x+dx) - f(x)}{dx}$$

```
#include <stdio.h>
#include <math.h>
double myfunction();

int main(void)
{
     double x,dx,A,B,deriv;
     printf("Enter x and dx \n");
     scanf("%lf %lf",&x,&dx);
     A=myfunction(x);
     B=myfunction(x+dx);
     deriv=(B-A)/dx;
     printf(" df/dx=  %lf  \n",deriv);
     return 0;
}


double myfunction(double x)
{
     double fofx;
     fofx=x*x;
     return fofx;
}
```

[PS2-1] What does the program in [3] give for $(x, dx) = (3.0, 0.1)$ and for $(x, dx) = (3.0, 0.01)$ and for $(x, dx) = (3.0, 0.001)$? Discuss this in terms of Eq. 1.

[PS2-2] Modify the program in [3] to compute the derivative of $\cos(x)$. Note: You are **not** allowed to use the known answer for df/dx in your code. You are only allowed to use f itself. You will need to link to the math library when compiling to use the cosine function. What does your program give for $(x, dx) = (0.0, 0.01)$ and for $(x, dx) = (1.57, 0.01)$? Explain why your answers are correct. (In explaining, you are allowed to use the known answer from your calculus courses.)

[PS2-3] Modify the program in [3] to compute the derivative of $f(x) = x^7 e^x \sin(x)$. Again: You are **not** allowed to use the known answer for df/dx in your code. What does your program give for $(x, dx) = (0.9, 0.001)$? Explain why your answers are correct. (In explaining, you are allowed to use the known answer from your calculus courses.)

[4] A code for integration! This code implements the usual definition of the integral as a Reimann sum using the rectangle rule. It integrates the function $f(x)$ from $x = a$ to $x = b$ by dividing the interval (a, b) into N segments of length $dx = (b - a)/N$.

```
#include <stdio.h>
#include <math.h>
double myfunction();

int main(void)
{
     int j,N;
     double a,b,x,dx,f,integral=0;
     printf("Enter a,b and N \n");
     scanf("%lf %lf %i",&a,&b,&N);
     dx=(b-a)/N;
     for (j=0;j<N;j=j+1)
     {
         x=a+j*dx;
         f=myfunction(x);
         integral=integral+f*dx;
     }
     printf(" integral=  %lf  \n",integral);
     return 0;
}


double myfunction(double x)
{
     double fofx;
     fofx=x*x;
     return fofx;
}
```

[PS2-4] Explain what the line "$x = a + j * dx$" is accomplishing in the code above. Explain what the line "integral=integral+$f * dx$" is accomplishing in the code above.

[PS2-5] Modify your code to implement the trapezoidal rule from your calculus course.

[PS2-6] Compare your answers for the integral for $a = 2, b = 5$ using the original rectangle rule code with $N = 10$ and $N = 100$ and using the trapezoidal rule with $N = 10$ and $N = 100$. What is the exact answer? Which rule works better? Why?