Modified Euler's Equation.

The Euler forward scheme may be very easy to implement but it can't give accurate solutions. A very small step size is required for any meaningful result. In this scheme, since, the starting point of each sub-interval is used to find the slope of the solution curve, the solution would be correct only if the function is linear. So an improvement over this is to take the arithmetic average of the slopes at x_i and x_{i+1}(that is, at the end points of each sub-interval). The scheme so obtained is called modified Euler's method. It works first by approximating a value to y_{i+1} and then improving it by making use of average slope.

$$y_{i+1} = y_i + h/2\,(y'_i + y'_{i+1})$$

$$= y_i + h/2(f(x_i, y_i) + f(x_{i+1}, y_{i+1})$$

$$y_{i+1} = y_i + h\,y'_i + h^2 y_i''/2 + h^3 y_i'''/3! + h^4 y_i^{iv}/4! + \ldots$$

$$f_{i+1} = y'_{i+1} = y'_i + h\,y''_i + h^2 y_{i'}'''/2 + h^3 y_i^{iv}/3! + h^4 y_i^{v}/4! + \ldots$$

By substituting these expansions in the Modified Euler formula gives

$$y_i + h\,y'_i + h^2 y_i''/2 + h^3 y_i'''/3! + h^4 y_i^{iv}/4! + \ldots = y_i + h/2\,(y'_i + y'_i + h\,y''_i + h^2 y_{i'}'''/2 + h^3 y_i^{iv}/3! + h^4 y_i^{v}/4! + \ldots)$$

Pseudo-code

```
void modified_euler()
{
        // Declarations.
        printf("\n Enter starting value of x & y i. e. x0, y0");
        scanf("%f%f",&x0, &y0);
        printf("\nEnter the value of x at which y is required ");
        scanf("%f", &xn);
        printf("\n Enter the step size ");
        scanf("%f",&h);
        printf("\n Enter the correction limit ");
        scanf("%f",&corr);
        do
```

```
        {
                printf("\nThe value of y = %f at x = %f", y0, x0);
                y1 = y3 + h*0.5*slope(x0, y0);
                diff=fabs(y-y1);
                y=y2;
                printf(" %0.3f %f",x+h,y1);
                do{
                        y1 = y3 + h*0.5*slope(x0, y0);
                        y1 = y3 + h*0.5*slope(x0, y0);
                        diff=fabs(y-y1);
                }while(diff*pow(10,corr)>1);
                x0 = x0 + h;
                y0 = y1;
        }while(x0<=xn);
}
```

Runge-Kutta 4th order

Let an initial value problem be specified as follows:

Here y is an unknown function (scalar or vector) of time *t*, which we would like to approximate; we are told that, the rate at which *y* changes, is a function of *t* and of *y* itself. At the initial time the corresponding *y* value is The function *f* and the data are given.

Now pick a step-size *h* > 0 and define

for *n* = 0, 1, 2, 3, ..., using

Here *y(n+1)* is the RK4 approximation of *y(tn+1)*, and the next value *y(n+1)* is determined by the present value *y(n)* plus the weighted average of four increments, where each increment is the product of the size of the interval, *h*, and an estimated slope specified by function *f* on the right-hand side of the differential equation.

- k1 is the increment based on the slope at the beginning of the interval, using y.
- k2 is the increment based on the slope at the midpoint of the interval, using k1.
- k3 is again the increment based on the slope at the midpoint, but now using k2.
- k4 is the increment based on the slope at the end of the interval, using k3.

Pseudo Code:

```
void runge_kutta()
{
// Declarations.
printf("\n Enter starting value of x & y i. e. x0, y0");
scanf("%f%f",&x0, &y0);
printf("\nEnter the value of x at which y is required ");
scanf("%f", &xn);
printf("\n Enter the step size ");
scanf("%f",&h);
do
{
k1 =  h*slope(x0, y0);
k2 = h*(slope((x0+h)/2,(y+k1)/2);

k3 = h*(slope((x0+h)/2,(y+k2)/2);

k4 = h*(slope((x0+h)/2,(y+k3));

k5=(k1 + 2*k2 + 2*k3 + k4)/6;
y0 = y0+k5;

x0=x0+h;
}while(x0<=xn);
}
```