**Assignment no.**

**Problem Statement: PROGRAM IN "c" TO DETERMINE TO SOLVE THE FOLLOWING 1ST ORDER DIFFERENTIAL**

**EQUATION USING "Runge Kutta 2nd Order Method". y' = f(x,y) = x^2 - y^3**

**Theory:**

In the forward Euler method, we used the information on the slope or the derivative of *y* at the given time step to extrapolate the solution to the next time-step. The LTE for the method is O(*h*2), resulting in a first order numerical technique. Runge-Kutta methods are a class of methods which judiciously uses the information on the 'slope' at more than one point to extrapolate the solution to the future time step. Let's discuss first the derivation of the second order RK method where the LTE is O(*h*3).

Given the IVP of Eq. 6, and a time step *h*, and the solution *yn* at the *n*th time step, let's say that we wish to compute *yn*+1 in the following fashion:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | *k*1 = *hf*(*yn*,*tn*) |  |
|  |  | $\displaystyle k_2 = hf(y_n+\beta k_1, t_n + \alpha h)$ |  |
|  |  | *yn*+1 = *yn* + *ak*1 + *bk*2, | (12) |

where the constants $\alpha$, $\beta$, *a* and *b* have to be evaluated so that the resulting method has a LTE O(*h*3). Note that if *k*2=0 and *a*=1, then Eq. 13 reduces to the forward Euler method.

Now, let's write down the Taylor series expansion of *y* in the neighborhood of *tn* correct to the *h*2 term i.e.,

|  |  |
| --- | --- |
| \begin{displaymath}y(t_{n+1}) = y(t_n) + h \frac{dy}{dt}\vert _{t_n} + \frac {h^2}{2} \frac{d^2y}{dt^2}\vert _{t_n} + {\mbox{O}}(h^3). \end{displaymath} | (13) |

However, we know from the IVP (Eq. 6) that *dy*/*dt* = *f*(*y*,*t*) so that

|  |  |
| --- | --- |
| \begin{displaymath}\frac{d^2y}{dt^2} = \frac{df(y,t)}{dt} = \frac{\partial f}{\p... ...rac{\partial f}{\partial t} + f \frac{\partial f}{\partial y}. \end{displaymath} | (14) |

So from the above analysis, i.e., Eqs. 14 and 15, we get

|  |  |
| --- | --- |
| \begin{displaymath}y_{n+1} = y_n + h f(y_n,t_n) + \frac{h^2}{2} \left[ \frac{\pa... ...ac{\partial f}{\partial y}\right](y_n,t_n) + {\mbox{O}}(h^3). \end{displaymath} | (15) |

However, the term *k*2 in the proposed RK method of Eq. 13 can be expanded correct to O(*h*3) as

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | $\displaystyle k_2 = hf(y_n+\beta k_1, t_n + \alpha h)$ |  |
|  |  | $\displaystyle = h\left( f(y_n,t_n) + \alpha h\frac{\partial f}{\partial t} (y_n... ... + \beta k_1 \frac{\partial f}{\partial t} (y_n, t_n)\right) + {\mbox{O}}(h^3).$ | (16) |

Now, substituting for *k*2 from Eq. 17 in Eq. 13, we get

|  |  |
| --- | --- |
| \begin{displaymath}y_{n+1} = y_n + (a+b) h f(y_n,t_n) + bh^2(\alpha \frac {\part... ...f \frac {\partial f}{\partial y})(y_n,t_n) + {\mbox{O}}(h^3). \end{displaymath} | (17) |

Comparing the terms with identical coefficients in Eqs. 16 and 18 gives us the following system of equations to determine the constants:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | *a*+*b*=1 |  |
|  |  | $\displaystyle \alpha b = \frac{1}{2}$ |  |
|  |  | $\displaystyle \beta b = \frac{1}{2}.$ | (18) |

There are infinitely many choices of *a*, *b*, $\alpha$ and $\beta$ which satisfy Eq. 19, we can choose for instance $\alpha = \beta = 1$ and *a*=*b*=1/2. With this choice, we have the classical second order accurate Runge-Kutta method (RK2) which is summarized as follows.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | *k*1 = *hf*(*yn*,*tn*) |  |
|  |  | *k*2 = *hf*(*yn*+*k*1, *tn* + *h*) |  |
|  |  | $\displaystyle y_{n+1} = y_n + (k_1 + k_2)/2, \:\:\: {\mbox{Second Order Runge-Kutta Method (RK2).}}$ | (19) |

**Variable Listing:**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Data Type** | **Purpose** |
| toFindX | double | For storing the value of x |
| X0, y0 | double | Stores primary value of x0 and y0 |
| k | double | Stores the stores the average of k1 and k2 |
| k1 | double | Stores the 1st formula |
| k2 | double | Stores the 2nd formula |
| h | double | Stores the height gap |
| nextY | double | Stores the next value of y |
| lastY | double | Stores the last value of y |

**Algorithm:**

1. Read the value of x0 and y0 from the user
2. Read toFindX and height gap h from the user
3. Store toFindX in x variable
4. Store y0 in lastY
5. Repeat through step 6 to step
6. Store dydx(x0, y0) \* h in k1 variable
7. Store dydx(x0 +h, y0 + k1) \* h in k2 variable
8. Store lastY + k in nextY
9. Store nextY in lastY
10. Store x0 + h in x0 variable
11. If x0 get equal or greater than toFindX, break the loop

[End of While Loop]

1. Display nextY to the user as output
2. End.

**Source Code:**

#include "stdio.h"

#define dydx(x, y) (x \* x + y \* y)

int main()

{

double toFindX, x0, h, k1 = 0, k2 = 0, y0, k = 0, nextY = 0, x, lastY;

printf("-----------------\n");

printf("Runge Kutta 2nd Order\n");

printf("-----------------\n");

printf("Enter the value for x0: ");

scanf("%lf", &x0);

printf("Enter the value for y0: ");

scanf("%lf", &y0);

printf("Enter the value for x to find: ");

scanf("%lf", &toFindX);

printf("Enter the value for h (height gap): ");

scanf("%lf", &h);

x = toFindX;

lastY = y0;

while (1)

{

k1 = h \* dydx(x0, y0);

k2 = h \* dydx(x0 + h, y0 + k1);

k = (k1 + k2) / 2;

nextY = lastY + k;

lastY = nextY;

x0 = x0 + h;

if (x0 >= toFindX)

break;

}

printf("Hence, the value of y(%lf) = %4.6lf", x, nextY);

return 0;

}

**Input/Output:**

-----------------

Runge Kutta 2nd Order

-----------------

Enter the value for x0: 0

Enter the value for y0: 1

Enter the value for x to find: 1

Enter the value for h (height gap): 0.2

Hence, the value of y(1.000000) = 2.708000

**Discussion:**

1. This program doesn’t run for a very large value.
2. The program exits when user puts a wrong interval.