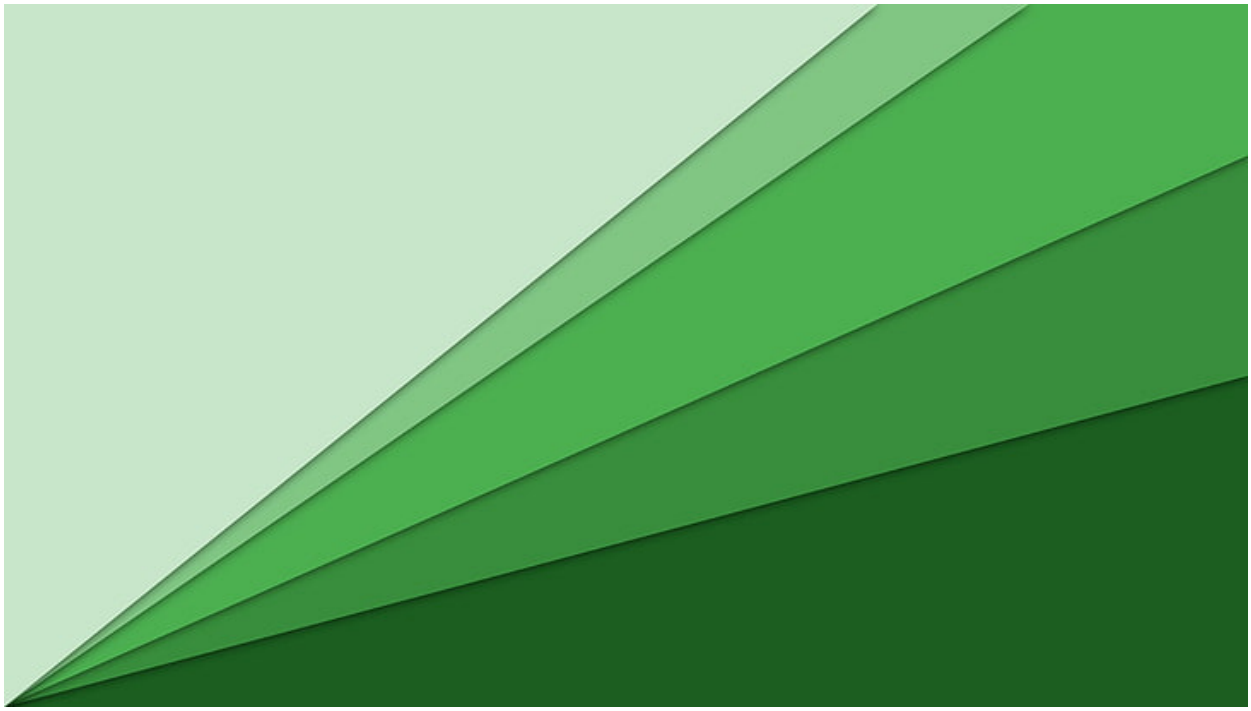


Linear ODE Approximator

12.14.2021

Yassin Kortam
Engineering 50
Abdollah Tabrizi

Abstract

The objective of this project is to create a C++ program to approximate the solution to the following initial value problem numerically:

$$\frac{d^2V}{dx^2} + 2\frac{dV}{dx} + 5V = 50 \quad V(0) = 0 \quad V'(0) = 0$$

In addition to this, the program conducts a simple error analysis given the exact analytic solution and exports relevant data into csv files for investigation. Note that f and V will be used interchangeably. Exceeding the stated objectives, this program can also solve any linear ordinary differential equation (LODE) with constant coefficients, and includes a class that handles user inputs to accommodate any independent variable name, order, or linear combinations of the derivatives of the independent variable.

The program successfully constructed a table of solution approximations for the given LODE, and error values, although initially large, rapidly diminished as both the exact solution and the approximation converged to the same value. A table expressing error as a function of step-size was also generated automatically, and depicted an interesting non-linear relationship between the two.

Derivations

This project investigates a computer program that solves linear ordinary differential equations (LODEs) of the general form:

$$\sum_{i=0}^{n} C_n f^{(n)}(x) = B$$

Where n is the order of the derivative of a function *f* of *x*, with *B* and all *Cs* being constants. The definition of a derivative, being:

$$f'(x) = \lim_{h \to 0} \frac{f(x) - f(x-h)}{h}$$

Is used to differentiate the function *f* at an arbitrary point x within a given interval. To solve a LODE of the general form given, the definition of a derivative substitutes all derivatives of *f,* and the resulting expression is solved in terms of multiples and variations of $f(x-kh)$, where k is a natural number, thereby isolating $f(x)$. However, to generalize this process for programming, an improvised version of the first derivative, which omits $f(x)$, is used. Let the following be called the "Yassin Derivative" (YD):

$$Y_d = \lim_{h \to 0}(\frac{-f(x-h)}{h})$$

The first and higher derivatives are to be taken in terms of Y_d and later corrected by adding the missing terms. This correction (YD correction), unlike the definition of a derivative of order n, can be easily expressed as a sum:

$$Y^{(n)}_{dcorrection} = \frac{C_n}{h^n} f(x) + \frac{C_n}{h^n} \sum_{i=1}^{k} (-1^k (n-k) f(x-kh)) \ , n > k$$

Since YD omits $f(x)$, it can be operated repeatedly within a for-loop without the need for algebraic corrections- the YD correction is arithmetic. Now, $f(x)$ can be expressed as the following:

$$f(x) = \frac{B - \sum_{i=1}^{n} C_n (Y_d^{(n)} + \frac{1}{h^n} \sum_{i=1}^{k} (-1^k (n-k) f(x-kh)))}{C_0 + \sum_{i=1}^{n} \frac{C_n}{h^n}}$$

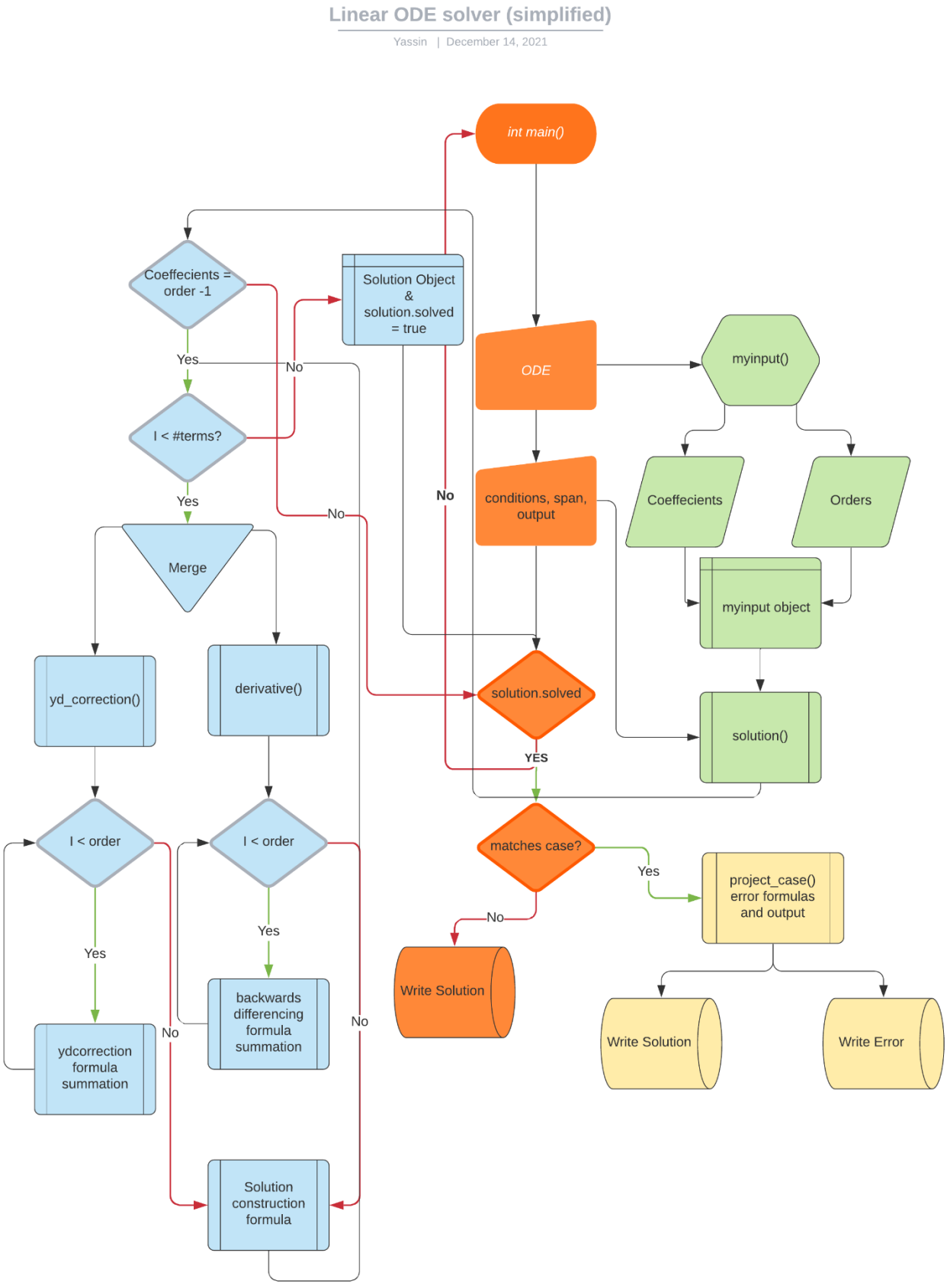
This rational function of sums can be simplified by expressing each component by the name of its corresponding function, and in practical terms this means creating functions in the program for each portion of the calculation before combining terms and conducting the divisions. YD correction can be split into the function YDR ($f(x)$ is omitted) and the constant YDL (portion of YD that is a multiple of $f(x)$), so $f(x)$ can be expressed as:

$$f(x) = \frac{B - \sum_{i=1}^{n} (C_n Y_d^{(n)}) - YDR}{C_0 + YDL}$$

Program Structure

Using the formulas derived and accounting for discontinuities, initial conditions, etc with vector/array indexing, functions were created to compute $f(x)$ for $0 \leq x < a$, $\frac{a}{h}$ times, stepping *x* by *h* after each iteration. The user inputs *a*, and *h* is a coded quantity (0.00001). Due to the complexities associated with creating a robust program that can solve the general LODE case given, the following flowchart is a simplification that summarizes the critical functions and logic of the code. The function yd_correction() is equivalent to YDR, derivative() is YD, and YDL is accounted for in the solution construction formula of the private member function of the solution class solve(). All variables are public and all member functions are private and are only accessed by the constructor.

Unfortunately, due to platform restrictions, the program does not create a plot automatically, but outputs a csv file that can be plotted with spreadsheet software.


Linear ODE solver (simplified)
Yassin | December 14, 2021
int main()
ODE
myinput()
Coeffecients
Orders
myinput object
conditions, span, output
solution()
Solution Object & solution.solved = true
Coeffecients = order -1
Yes
No
I < #terms?
Yes
No
Merge
yd_correction()
derivative()
solution.solved
YES
No
I < order
I < order
Yes
Yes
ydcorrection formula summation
backwards differencing formula summation
No
No
Solution construction formula
matches case?
Yes
No
Write Solution
project_case() error formulas and output
Write Solution
Write Error

User Experience

1. The user enters any differential equation of the form $\sum_{i=0}^{n} C_n f^{(n)}(x) = B$
2. The coefficients and orders are outputted in their corresponding orders to confirm their correctness.
3. The program asks for the minimum number (order) of initial conditions for $f(x)$ to run. In the given case since h is small and $V'(0) = 0$, it can be approximated that $f(0)$ = $f(h)$ = 0. Assumptions like this may negatively impact error for other functions.
4. The user is prompted to enter the maximum value a for the solution window $0 \leq x < a$.
5. The user enters the output data file path and/or name.
6. The program outputs a table showing the approximated values of *f* for all integer values in $0 \leq x < a$, along with the minimum and maximum values of *f* with corresponding times t (x=t).
7. If the program detects the LINODE given for this project (matches case), the user is prompted to enter the number of terms calculated for the function error of h, and the output file path and/or name;

Demo

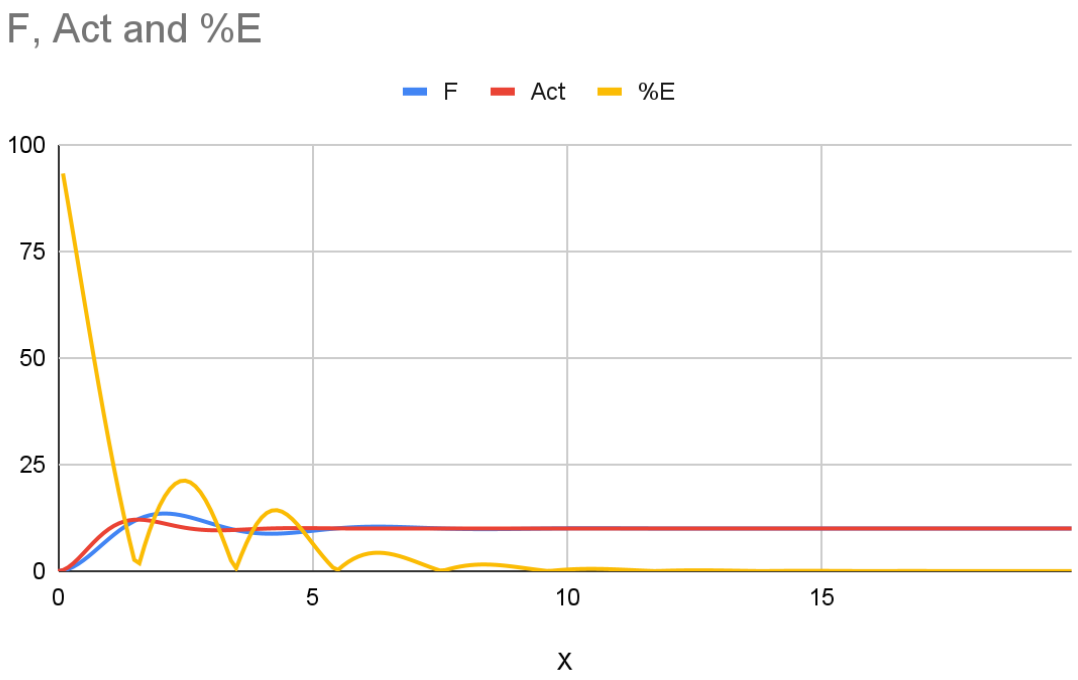
```
********** Inputs **********

Enter a linear nth order differential equation in the form Cnvariable(order) +....= B
v(2) + 2v(1) + 5v(0) = 50


*****************************
Coefficients:
1
2
5
50

*****************************
Orders:
2
1
0
```

```
Enter initial conditions (roughly equivalent for small h):
v(0) = 0
v(1e-05) = 0

What should the solution span?
0<x<15

Enter output location:
/Users/yassinkortam/Desktop/mysolution.csv

********* Simple Solution *********

      x       v(x)
      v(0)       0
      v(1)   7.55404
      v(2)   13.4688
      v(3)   11.1975
      v(4)   8.82672
      v(5)   9.45878
      v(6)   10.3852
      v(7)   10.2322
      v(8)   9.87823
      v(9)   9.90415
      v(10)   10.0366
      v(11)   10.0384
      v(12)   9.98984
      v(13)    9.985
      v(14)   10.0024

******************
 Max: 13.5091     t = 2.09441
 Min: 0     t = 0

*******Error Analysis********

How many terms would you like?
1000

Enter output location:
/Users/yassinkortam/Desktop/myerror.csv
Program ended with exit code: 0
```

Data

F, Approximation, and Error as Functions of x

For a fixed step-size h, the error decreases nonlinearly in an oscillating pattern as *x* increases and *f* and its approximation converge asymptotically. The graph below, F is the approximation, Act is the exact analytic solution *f*, and %E is the absolute value of the difference multiplied by one hundred divided by the approximation.

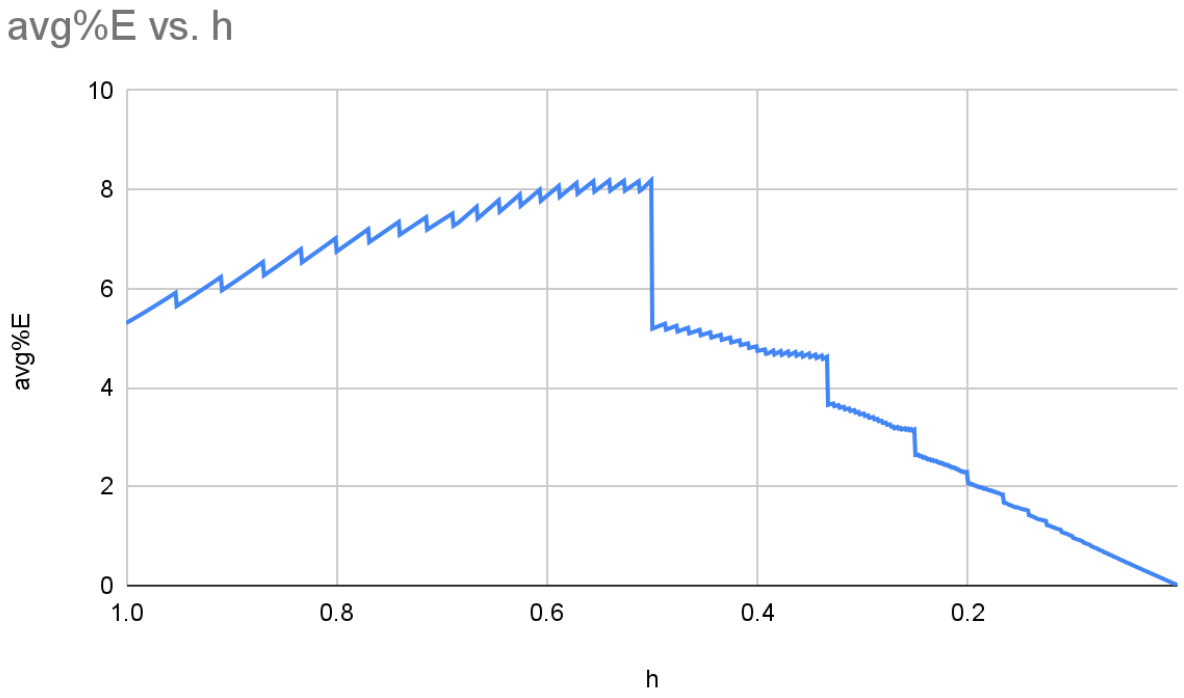


X	F	Act	%E
0	0	0	nan
1	7.55414	9.85836	30.5028
2	13.4688	11.3967	15.3842
3	11.1975	9.59152	14.342
4	8.82671	9.93605	12.5679
5	9.4588	10.0749	6.51317
6	10.3852	9.98573	3.84622
7	10.2321	9.99424	2.32515
8	9.87823	10.0037	1.2701

9	9.90415	9.99965	0.964231
10	10.0366	9.99961	0.368269
11	10.0384	10.0002	0.380855
12	9.98984	10	0.101685
13	9.985	9.99998	0.149963
14	10.0024	10	0.0244231
15	10.0057	10	0.057235
16	9.99959	10	0.00407764
17	9.99786	10	0.0214223
18	9.99997	10	0.000327648
19	10.0008	10	0.00785622

Error as a Function of h

For a variable step-size h incremented by 0.001, the error function exhibits nonlinear behavior, increasing and then rapidly decreasing as h approaches zero. This behavior might differ depending on the differential equation. The error function modeled is the average percent error of the approximation over the interval modeled.



h	avg%E
1	5.30221
0.9	6.09135
0.8	6.74766
0.7	7.37894
0.6	7.87554
0.5	5.18954
0.4	4.7397
0.3	3.47021
0.2	2.07081
0.1	0.962092
0.001	0.00943714

Conclusions

I. Error decreases as x increases

This is not a rule for all cases, it happens that the given differential equation has a horizontal asymptote at V = 10, and the approximation modeled the analytic solution to a satisfactory degree.

II. Error of h decreases as h decreases

More experimentation and analysis is needed to determine if the graph of this function is dependent on the differential equation, but for this case it increases briefly before plummeting as h approaches 0.

Source Code