International Journal of Scientific Engineering and Applied Science (IJSEAS) – Volume-2, Issue-6,June 2016
ISSN: 2395-3470
www.ijseas.com

Efficient algorithms HCM and SM to solving system of non-linear equations using animation techniques

ASHOKA S. B., Research Scholar
Department of Computer Science and Applications
Bangalore University, Bangalore – 560 009, India

Abstract

The standardize the measurement of algorithm efficiency depends an algorithm is a systematic method containing a sequence of instructions to solve a computational problem. It takes some inputs, performs a well defined sequence of steps, and produces some output. Once we design an algorithm, we need to know how well it performs on any input. A major criterion for a good algorithm is its efficiency that is, how much time and memory are required to solve a particular problem. Intuitively, time and memory can be measured in real units such as seconds and megabytes. However, these measurements are not subjective for comparisons between algorithms, because they depend on the computing power of the specific machine and on the specific data set. Algorithm can be written in pseudo-code because simplification of the actual implementation is very good way. In homotopy continuation method algorithm for solving system of non linear algebraic equation, given the generalized uniform approach of three more problems result. We comparison HCM results with other different methods algorithms observations like Shooting method[RK45]. The detailed observations of all the methods of algorithm noted and all algorithm yields unique approach on the results with limiting conditions. But our one of proposed works HCM algorithm given a excellent results, for all set of parameters with high end values. One more highlight of this chapter is obtained the solution also represented through 3D visualization using open source code Mayavi version 0.6.

Keywords: Homotopy Continuation[HCM], Shooting[SM], DBF, RKF45.

Introduction

The exact speed of an algorithm depends on where the algorithm is run, as well as the exact details of its implementation, computer scientists typically talk about the runtime relative to the size of the input. The time complexity of an algorithm is commonly expressed using big O notation, which excludes coefficients and lower order terms. When expressed this way, the time complexity is said to be described asymptotically, i.e., as the input size goes to infinity. For some optimization problems, we can reach an improved time complexity, but it seems that we have to pay for this with an exponential space complexity. Note that algorithms with exponential space complexities are absolutely useless for real life applications. Theoretical computer science has its uses and applications and can turn out to be quite practical. In this article, targeted at programmers who know their art but who don't have any theoretical computer science background, I will present one of the most pragmatic tools of computer science: Big O notation and algorithm complexity analysis. In general, time complexity is considered much more important than space complexity, in part because the memory requirement of most algorithms is lower

than the capacity of current machines. In the rest of the section, all calculations and comparisons of algorithm efficiency refer to time complexity as complexity unless otherwise specified. Also, time complexity and running time can be used interchangeably in most of the cases. The time complexity of an algorithm is calculated on the basis of the number of required elementary computational steps that are interpreted as a function of the input size. Most of the time, because of the presence of conditional constructs (e.g., if-else statements) in an algorithm, the number of necessary steps differs from input to input. Thus, average-case complexity should be a more meaningful characterization of the algorithm. However, its calculations are often difficult and complicated, which necessitates the use of a worst-case complexity metric. An algorithm's worst-case complexity is its complexity with respect to the worst possible inputs, which gives an upper bound on the average-case complexity. As we shall see, the worst-case complexity may sometimes provide a decent approximation of the average-case complexity. The theory of computational complexity was developed Ullman (1984) Papadimitriou (1993, 1998), Wilf (2002). This allows an algorithm's efficiency to be estimated and expressed conceptually as a mathematical function of its input size. Generally speaking, the input size of an algorithm refers to the number of items in the input data set. In this chapter we will try to found the computational complexity of our generalized module of HCM with some standard results.

//Algorithm of Homotopy Continuation Method//

Main module

//Input: (Initial un-known values)//

//Output: (Refined solution of the method)//

Step 1: Start

Step2: Initialization of matrices

i) Initialization of x[n]

ii) Initialization of final matrix final_k[n][n]

iii) Initialization of previous matrix prev[n]

iv) Initialization w_k[n]

Calculation of the final K values

Step 3: Differential equation (module 1)

diff_eq(n),Parameters:mat, x[0],x[1],…..x[n],dim

Step 4: Inverse equation (module 2)

inverse(n);

Parameters: mat, dim

Step 5: Normal equation (module 3)

nrml_eq(n);

Calculation of Final matrix: multiplication(n)

Parameters: invmat, consteq, dim

Step 6: Calculation of Sum of K values

$$K_1 = f(t_n, y_n),$$

$$K_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1),$$

$$K_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2),$$

$$K_4 = f(t_n + h, y_n + hK_3).$$

Where n = 0, 1, 2, 3, . . .

k_1 is the increment based on the slope at the beginning of the interval, using $\dot{y}$, (Euler's method) ;

k_2 is the increment based on the slope at the midpoint of the interval, using $\dot{y} + \frac{h}{2}k_1$;

k_3 is again the increment based on the slope at the midpoint, but now using $\dot{y} + \frac{h}{2}k_2$;

k_4 is the increment based on the slope at the end of the interval, using $\dot{y} + hk_3$.

Step7: Calculate

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$t_{n+1} = t_n + h$$

Now pick a step-size $h > 0$

Step 8: Calculation of W values
Step 9: Print Refined values.
Step10: Stop.

Normal matrix module

//Input**:** **(**write input to the module**)**//
//Output**:** **(**write output from the module**)**//
Step 1: Start
Step 2: Parameters: mat, dim ,n,N.
Step3:Intialization Normal_Matrixa[n][n]
Initialization parameter matrices gn[n],gdas[n], yn[n], esp1[n], gnsqr[n], bkr=1.2;
Step 4: for n=1 DO n
Calculate:

$$y_n(n) = \varepsilon + n\frac{(1-\varepsilon)}{N}$$
$$G(y_n) = e^{\frac{\log(1+v)}{1-\varepsilon}(y_n-\varepsilon)}$$
$$G'(y_n) = \frac{\log(1+v)}{1-\varepsilon} e^{\frac{\log(1+v)}{1-\varepsilon}(y_n-\varepsilon)}$$

where $\varepsilon = R_2 R_1^{-1}$ (ratio of inner to outer cylinder radii).
v variable viscosity.
Step5:
Calculate:

$$f_n(\underline{U}) = BY_n G_n^2 U_n^2 - \left(\frac{N^2}{(1-\varepsilon)^2} Y_n G_n + N\frac{(\delta G_n - Y_n G_n^1)}{2(1-\varepsilon)}\right) U_{n+1}$$
$$+\left(2\frac{N^2}{(1-\varepsilon)^2} + A\right) Y_n G_n U_n -$$
$$\left(\frac{N^2}{(1-\varepsilon)^2} Y_n G_n - N\frac{(\delta G_n - Y_n G_n^1)}{2(1-\varepsilon)}\right) U_{n-1} - \Lambda Y_n G_n^2.$$

Step6: Return Functional values.
Step8: Stop.

Differential matrix module

Step1: Start.
Step2: Accept A, B, N, v, δ, matrix (set of parameters received**)**
Step3: For I = 1 DO N
Step4: Calculate:

$$esp1[n] = pow((1-\varepsilon), 2)$$
$$Y_n = \varepsilon + n\frac{(1-\varepsilon)}{N}$$
$$G_n = e^{\frac{\log(1+v)}{1-\varepsilon}(y_n-\varepsilon)}$$
$$G_n^2 = e^{\frac{\log(1+v)}{1-\varepsilon}(y_n-\varepsilon)}$$
$$G_n^1 = \frac{\log(1+v)}{1-\varepsilon} e^{\frac{\log(1+v)}{1-\varepsilon}(y_n-\varepsilon)}$$

where $\varepsilon = R_2 R_1^{-1}$ (ratio of inner to outer cylinder radii).
V variable viscosity**.**
Step5:
Calculate:

$$f_n(\underline{U}) = 2*BY_n G_n^2 U_n + \left(\frac{N^2}{(1-\varepsilon)^2} Y_n G_n + N\frac{(\delta G_n - Y_n G_n^1)}{2(1-\varepsilon)}\right) U_{n+1}$$
$$+\left(2\frac{N^2}{(1-\varepsilon)^2} + A\right) Y_n G_n U_n - \left(\frac{N^2}{(1-\varepsilon)^2} Y_n G_n - N\frac{(\delta G_n - Y_n G_n^1)}{2(1-\varepsilon)}\right) U_{n-1} - \Lambda Y_n G_n^2.$$

Step6: Return Functional values.
Step7 : Stop.

Inverse matrix module

//Input: **(**write input to the module**)**//
//Output: **(**write output from the module**)**//
Step 1: Start
Step 2: Parameters: mat, dim ,n
Step 3: Intialization of matrix a[n][n]
Intialization of matrix b[n][n]
Step 4: Calculation of a[i][j]
Calculation of b[i][j]
Step 5: if i==k
r=a[i][k]/a[k][k]
a[i][j]=a[i][j]-r*a[k][j]
b[i][j]=b[i][j]-r*b[k][j]
Step 6: Matrix Multiplication
Parameters: invmat, consteq, n1
Initialization of matrices- cmat, b[n][n], inv[n][n], c[n][n]

Calculation of matrices - inv[i][j], b[i][j]
Step 7: Calculation of matrix cmat
Step 8**:** Return the result cmat
Step 9**:** Stop

Multiplication matrix module

Step1: Start.
Step2:Accept N,h, Inverse_Matrix[n][n], Constant_Matrix[n][n].
Step3: Calculate
Step4: For I = 0 DO N
Step5: For J = 0 DO N
Step6:Multi_Matrix[n][n]=h*(Inverse_Matrix[n][n]* Constant_Matrix[n][n])
Step9: Return Functional values of Multi_Matrix[n][n].
Step10: Stop.

Results and Discussions

Comparisons between HCM and Shooting Method of solving DBF flow through an Rectangular porous channel, Cylindrical Porous Annulus

Method of Evaluation	Fixed values of Da=5 F=10,BRK=1.2	Fixed values of Da=10 F=10,BRK=1.2	Fixed values of Da=20 F=10,BRK=1.2
SHM	0.039296897	0.009989651	0.002499844
HCM	0.037789136	0.009522840	0.002380943

Table 1: Difference between Homotopy and shooting method

Methods	HCM	SM
Fixed values of parameter	Da=10, esp=0.02, h=0.1	Da=10,esp=0.02 , h=(1-esp)/N
Results	0.01166	0.01044

Table 2: Difference between Homotopy and shooting method

Diff = 0.0116- 0.0104 = 0.0012

For Different Values of Epsolen Da=5, Re=10, Cf=0.05, Lamda=1				
Eps=1.0	Eps=0.2	Eps=0.3	Eps=0.02	Eps=0.01
0.034461792	0.031965518	0.029035028	0.050987731	0.1826333091

Table 3: HCM result comparisons for different Epsolen

Different Da Values	HCM	Shooting method	Differences
05	0.0377	0.0395	0.0017
10	0.0095	0.0099	0.0004
20	0.0023	0.0024	0.0001

Table 4(a): Difference between Homotopy and Shooting Method Fixed F=10, BRK=1.2

Diff. Brinkmen	HCM	SM	Diff.
0.8	0.0095	0.0099	0.0004
1.0	0.0095	0.0099	0.0004
1.2	0.0095	0.00999	0.0004

Table 4(b): Difference between Homotopy and Shooting Method Fixed F=10, Da=10

HCM result visualization in 3D Simulation

3D simulation software is used to present Darcy-Brikman-Forchheimer flow through annulus. Simulation gives visual sequence of different set of parameter in pours media.

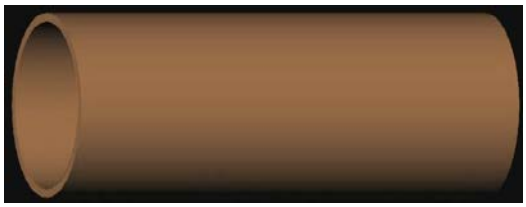
POROUS MEDIA MODELLING.

Maya polygon is used to create 3 dimensional porous media.

Step1:Create cylinder

1.Create a NURBS cylinder and name it "*c_i*", for "inferior constraint". Go to the Channel Box and set its Y scale to 10, and its Z scale to 0.4.

2. Enter "Insert" to edit the Pivot Point position, and then go to the Numerical Input Line, which is a white space just above the Channel Box. Enter the values 0 - 10 0.

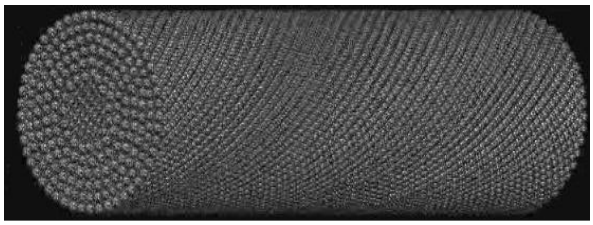


Step 2:

Create cylinder with porous media.

1.Create Polygonal sphere and name it "porous Particle"

1. Duplicate porous particles to fill inside the cylinder.



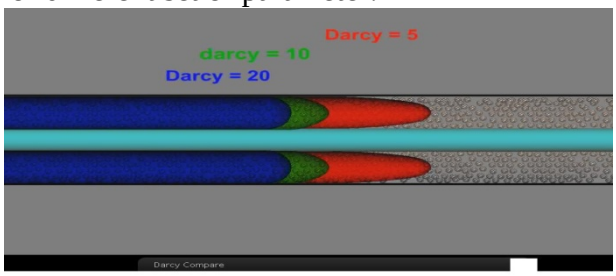
Step 3:

Create annulus with porous media

Fill the porous particles centered with iron rod inside the cylinder created in step 1.

Animation:

Create expression to animate velocity profile for different set of parameter.



References

[1] D. A. Nield and A. Bejan, Convection in porous media, Springer Verlag, New York, 2006.

[2] K. Vafai, Hand book of porous media, CRC Press, 2005.

[3] N. Rudraiah, P. G. Siddheshwar, D. Pal, and D. Vortmeyer, Non – Darcy effects on transient dispersion in porous media, ASME Proc. 1988, Nat. Heat Trans. Conf., Houston, Texas, USA (Ed. H. R. Jacobs), HTD – 96 (1) (1988) 623 - 629.

[4] E. Skjetne and J. L. Auriault, New insights on steady, non-linear flow in porous media, Eur. J. Mech. B/Fluids, 18 (1999) 131-145.