Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 45 No. 2 (2024)

Multicolored Approaches for Ordinary Differential Equation using Scilab

Dr.Revathi.R.[1*], Gayathri.S.[2], Gayathri.R.[3], B.Arun[4]

[1,2] Assistant Professor,Department of Computer Science,Karpagam Academy of Higher Education,Coimbatore,India.*

[3] Assistant Professor,Department of Mathematics,Karpagam Academy of Higher Education,Coimbatore,India

[4] Assistant Professor,Department of Science and Hummanities,,Karpagam Academy of Higher Education,Coimbatore,India

***Abstract*—** Computer knowledge and mathematics are coupled to each other in one way or the other as extreme of the computer programs has its base from numerous fine propositions and fields but these aren't abecedarian to it. In this paper we're going to deal with an operation called ODE which has its base on fine function of one independent variable and the outgrowth of those function. ODE is an inbuilt function in Scilab, which makes our programming easier, we only have to mention the keyword with original or boundary value. In this paper we will readily bandy the basics of ODE in Scilab. This paper originally starts with the detailed modeling an ODE with three different styles(in which standard Scilab programming

***Keywords*—**ODE,Scilab

I. Introduction To Ode In Scilab

Scilab is a free and open-source, cross-platform numerical calculation package renowned for its extensive collection of numerical algorithms catering to various scientific computing problems. It functions as an interpreted programming language, enabling rapid development due to its integration with advanced programming languages and feature-rich libraries.

Scilab allows users to develop their own modules, fostering a community-driven ecosystem. It stands as a prominent alternative to MATLAB in the realm of open-source numerical computation, alongside GNU Octave. While Scilab's syntax is akin to MATLAB, it places less emphasis on strict compatibility, differing from Octave in this aspect. However, Scilab provides a source code translator to facilitate the conversion of MATLAB code to Scilab, easing the transition for users.

Executing Scilab code is straightforward, typically done by entering it directly into the graphical command window prompt. The software is freely available under an open-source license, and contributions from users are integrated into the main program, reflecting its collaborative development model.

Scilab is particularly tailored for precise calculations, boasting a wide range of capabilities. Some of its key features include:

- Numerical algorithms for diverse scientific computing tasks.
- Graphical plotting and visualization tools.
- Symbolic mathematics capabilities.
- Matrix operations and linear algebra functionalities.
- Statistical analysis and data processing tools.

Overall, Scilab serves as a powerful platform for scientific computing, offering an accessible and versatile environment for numerical analysis and simulation tasks.

Scilab encompasses a broad spectrum of computational capabilities, including linear algebra operations, manipulation of sparse matrices, polynomial and rational function interpolation and approximation, solvers for ordinary and algebraic differential equations, control systems analysis and design (both classic and robust control), optimization through direct matrix inequality techniques, as well as tools for signal processing and statistical analysis.

A. Ordinary Differential Equation-ODE

The Ordinary Differential Equation (ODE) solver in Scilab is commonly utilized with various input and output arguments. It solves explicit ordinary differential equations of the form:

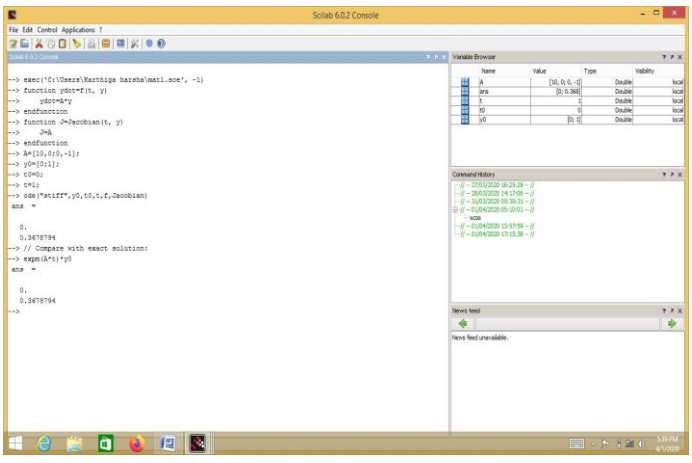
$dy/dt=f(t,y),y(t0)=y0$
$0dtdy=f(t,y),y(t0)=y0$

In the given example, we aim to solve the equation $dt/dy=A\cdot y$, where the exact solution is y(t)=expm(A·t)·y0, with expm representing the matrix exponential.

The unknown in this case is the 2-by-1 matrix $y(t)$.

```
// Function defining the derivative dy/dt = f(t, y)

function ydot = f(t, y)

    A = [10, 0; 0, -1];

    ydot = A * y;

end function

// Function defining the Jacobian matrix J = df/dy

function J = Jacobian(t, y)

    A = [10, 0; 0, -1];

    J = A;

End function

 // Initial conditions and time span

A = [10, 0; 0, -1];

y0 = [0; 1];

t0 = 0;

t = 1;

// Solving the stiff ODE using the provided functions

ode("stiff", y0, t0, t, f, Jacobian)
```

This script sets up the system with the given matrix A, initial conditions y_0 and time span t_0 to t. It then employs the `ode` function with the `'stiff'` option to solve the differential equation system using the provided functions



for f(t,y) and f(t, y).

Fig 1:Ordinary Differntial Equation-ODE

In this example, we aim to solve the ordinary differential equation dx/dt=A·x(t)+B·u(t), where u(t)=sin(ω·t). Additional arguments A, u, B, and ω are passed to the function f within a list.

```
// Function defining the derivative dx/dt = f(t, x, A, u, B, omega)
function xdot = linear(t, x, A, u, B, omega)
    xdot = A * x + B * u(t, omega);
end function
// Function defining the input function u(t, omega)
function ut = u(t, omega)
    ut = sin(omega * t);
end function
// Initial conditions and parameters
A = [1, 1; 0, 2];
B = [1; 1];
omega = 5;
y0 = [1; 0];
t0 = 0;
t = [0.1, 0.2, 0.5, 1];
// Solving the ODE using the provided functions and parameters
ode(y0, t0, t, list(linear, A, u, B, omega))
```

This script sets up the system with the given matrices A and B, the function u(t,ω), initial conditions y_0 , and time span t_0 to t.It then employs the **'ode'** function to solve the differential equation system using the provided functions and parameters.

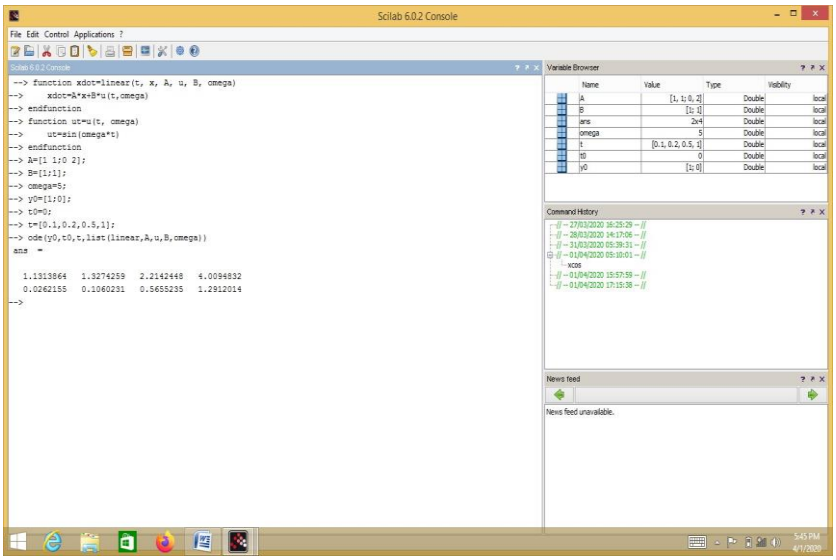


Fig 2:Ordinary Differntial Equation-Linear

In this example, we aim to solve the ordinary differential equation dt/dY =A·Y, where the unknown Y(t) is a 2-by-2 matrix. The exact solution is Y(t)=expm(A·t), where expm represents the matrix exponential.

```
    // Function defining the derivative dY/dt = f(t, y, A)
function ydot = f(t, y, A)
    ydot = A * y;
end function
```

```
// Initial conditions and parameters
A = [1, 1; 0, 2];
y0 = eye(A); // Initial condition Y0 as an identity matrix of A
t0 = 0;
t = 1;

// Solving the ODE using the provided function and parameter
ode(y0, t0, t, list(f, A))
```

This script sets up the system with the given matrix A, initial conditions Y_0 as the identity matrix of A, and time span t_0 to t. It then employs the ode function to solve the differential equation system using the provided function and parameter.

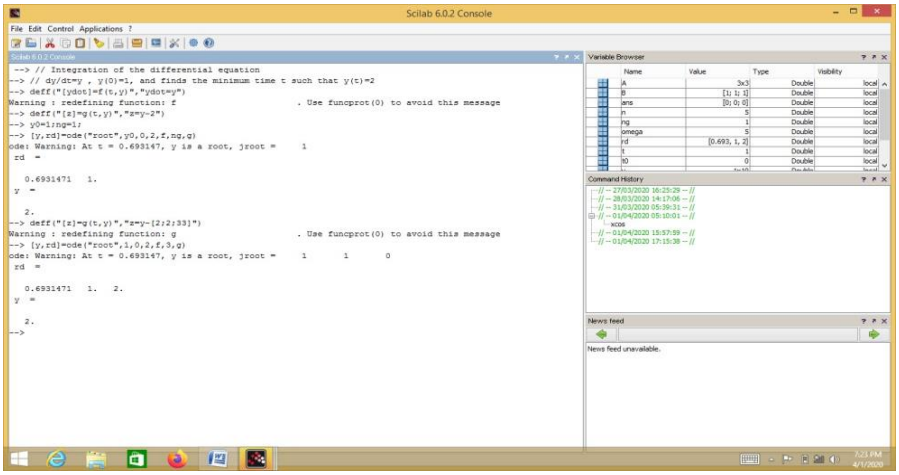


Fig 3: Ordinary Differntial Equation

// Compare with the exact solution:

expm(A*t) ode("adams",y0,t0,t,f)[2]

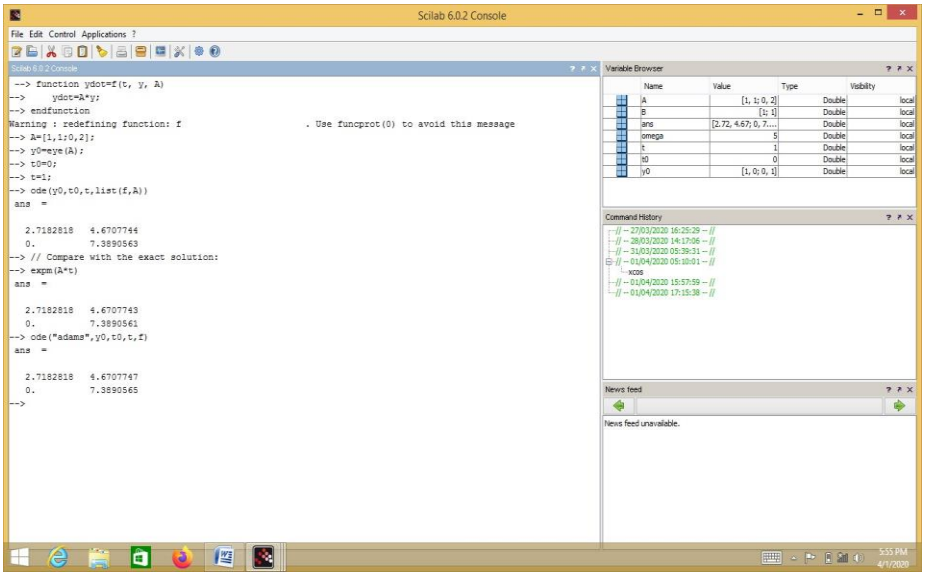


Fig 4:Ordinary Differntial Equation-Matrix

B. ***ODE_Discrete***

Ordinary differential equation solver, discrete time simulation

Syntax
y=ode ("discrete", y0, k0, kvect, f)
Arguments
Y0: a real vector or matrix (initial conditions).
T0: a real scalar (initial time).
F: An external i.e. function or character string or list.
K0: An integer (initial value)
Kvect: An integer vector.

An example of ODE_discrete:
y1=[1;2;3]; deff("yp=a_function(k,y)","yp=A*y+B*u(k)")
A=diag([0.2,0.5,0.9]); B=[1;1;1];u=1:10;n=5;
y=ode("discrete",y1,1,1:n,a_function); y (:,2)-(A*y1+B*u(1))

// Now y evaluates at [y3, y5, y7, y9]
y=ode ("discrete", y1, 1, 3:2:9, a_function) [2]

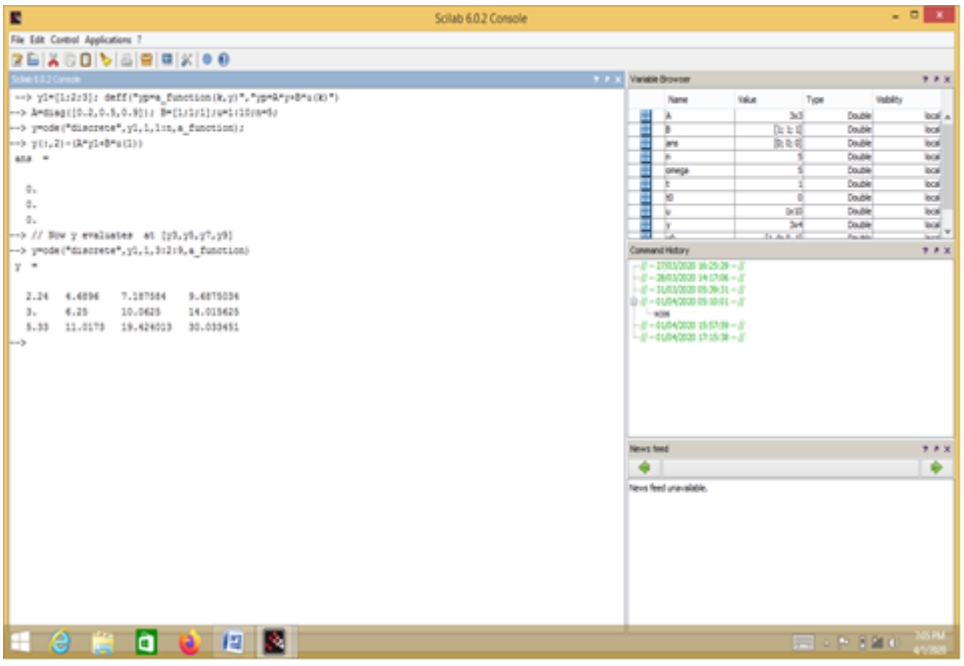


Fig 5:ODE-Discrete

C. ***ODE_Root:***

It seems you're describing a function ode that solves ordinary differential equations (ODEs) while also finding roots. Here's a revised version of the description for clarity:
Function: ode ("root", y0, t0, t [, rtol [, atol]], f [, jac], ng, g [, w, iw])

Description:
Solves ordinary differential equations (ODEs) while simultaneously finding roots of specified functions.
Arguments:

- y0: Initial conditions, a real vector or matrix.
- t0: Initial time, a real scalar.
- t: Times at which the solution is computed, a real vector.
- f: ODE function specified as an external function, character string, or list.
- rtol, atol: Relative and absolute tolerances, real constants, or vectors matching the size of y.
- jac: Jacobian of the ODE function, specified as an external function, character string, or list.
- ng: Number of functions to find roots for.
- g: Root functions specified as an external function, character string, or list.
- w, iw: Vectors of real numbers used internally.

Returns:
- y: Solution to the ODEs at the specified times, a real vector or matrix.
- rd: Roots found during the integration, a real vector.
- w, iw: Internal vectors of real numbers used during computation.

An example using ODE_root:

```
// Integration of the differential equation
// dy/dt=y , y(0)=1, and finds the minimum time t such that y(t)=2 deff("[ydot]=f(t,y)","ydot=y")
deff("[z]=g(t,y)","z=y-2")

y0=1;ng=1;
 [y,rd]=ode("root",y0,0,2,f,ng,g)
deff("[z]=g(t,y)","z=y-[2;2;33]")
[y,rd]=ode("root",1,0,2,f,3,g)
```
[2]

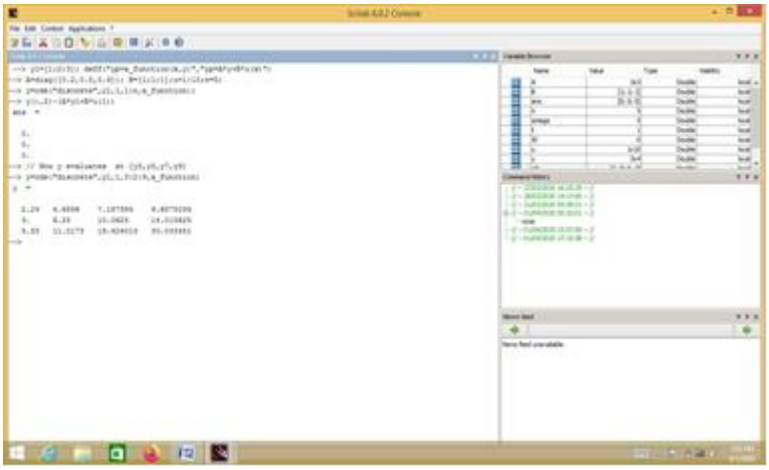


Fig 6:ODE-Root

D. **ODEDC**

Function: odedc(y0, nd, stdel, t0, t, f)

Description:
Discrete/continuous ODE solver that integrates a system of ordinary differential equations (ODEs) containing both continuous and discrete states, and potentially discontinuous changes.

Arguments:
- y0 : Initial conditions, a real column vector [Y0C; Y0D], where Y0C is a real vector representing continuous states, and Y0D is a real vector with nd components representing discrete states.
- nd : Number of components in the vector representing discrete states.

- stdel : Vector with one or two entries [h, delta], where h is the step size for integration and delta is the tolerance for detecting discontinuous changes (default value is 0).
- t0 : Initial time, a real scalar.
- t : Vector of time instants at which the solution is calculated, a real row vector.
- f : ODE function specified as an external function or character string with syntax:

yp = f(t, yc, yd, flag)

- If specified as a list, the syntax is:

List(f, p1, p2, ...)

Where the function f returns the function value as a function of (t, yc, yd, flag, p1, p2, ...), and p1, p2, ... are function parameters.

- If specified as a character string, it refers to the name of a C or Fortran routine. The Fortran calling sequence must be:

<f_name>(iflag, nc, nd, t, y, ydp)

double precision t, y(*), ydp(*)
integer iflag, nc, nd

And the C syntax must be:

void <f_name> (int *iflag, int *nc, int *nd, double *t, double *y, double *ydp)

Where:
- iflag = 0 or 1
- nc = number of continuous states yc
- nd = number of discrete states yd
- t = time
- y = [yc; yd; param], where param may be used to pass extra arguments specified in the ododc call.

Returns:
- yt : Solution to the system of ODEs at the specified time instants, a real matrix where each row corresponds to a time instant in t.

An example of ODEDC function is displayed below:

```
//Linear system with switching input deff('xdu=phis(t,x,u,flag)','if flag==0 then xdu=A*x+B*u; else
xdu=1-u;end');
x0=[1;1];
A=[-1,2;-2,-1]; B=[1;2];
u=0;
nu=1; stdel=[1,0]; u0=0; t=0:0.05:10;
xu=odedc([x0;u0],nu,stdel,0,t,phis); x=xu(1:2,:);
u=xu(3,:); nx=2;
plot2d(t, x', [1:nx], '161') plot2d2('onn',t',u',[nx+1:nx+nu],'000');
```

//Fortran external (see fydot2.f):
norm(xu-odedc([x0;u0],nu,stdel,0,t,'phis'),1)[9}

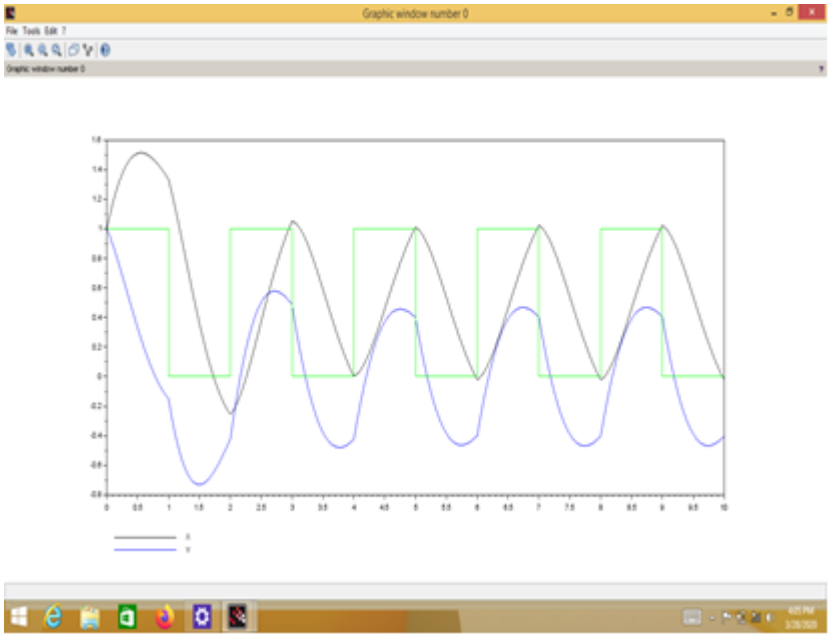


Fig 7:ODEDC

E. Application of ODE In Different Fields:

In Scilab, ordinary differential equations (ODEs) are employed to model various systems, including mechanical systems, electric circuits, the Van der Pol equation, the Rössler flow, oscillations in electric circuits, and predator-prey systems, among others. [1][3]

F. Weight Reduction Model

- Weight of person =xkg
- Tries to reduce weight
- Weight loss per month=10%
- Starting weight =100k

Dx/dt=-0.1x
Initial condition: X=100 at t=0
Cross multiplying we get : Dx/dt=-0.1x
Dx/x=-0.1dt
Integrating both sides we get Inx(t)/100=-0.

1t Compute and plot for 2years that is 24 months:
t=0:0.1:24
plot2d(t,100*exp(-0.1*t)); xtitle('weight_vs.month','time_in_months"weight _in_kg');[3]

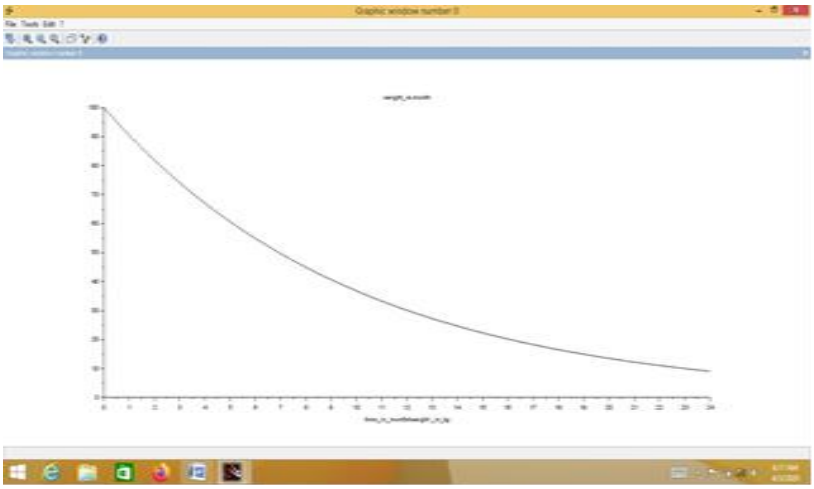


Fig 8: Weight Reduction Model

Conclusion :

The significance of Ordinary Differential Equations (ODEs) in Scilab cannot be overstated. This paper aims to provide an introductory overview of the ODE function in Scilab, explaining its syntax, demonstrating how to model ODEs in three different styles, and discussing various built-in ODE functions available in Scilab.

The core conclusion drawn from this approach is that while Scilab has the potential to guide users effectively through ODE modeling, it requires consistent and dedicated development to fully realize this potential.

Furthermore, Scilab excels in handling diverse types of ordinary differential equations and their systems commonly encountered in various industries. An important advantage is that Scilab is free of cost and open-source, allowing for seamless usage and occasionally demonstrating faster computation times compared to other software packages.

References

[1] E.balaguruswamy,"scilab textbook companion for numerical methods"s.pal,"scilab textbook companion for numerical methods :principles ,analysis and algorithms",ISBN:9780195693

[2] Manas Sharma,phd researcher at fridrich-schiller university,www.bragitoff.com

[3] Kannan m.moudgalya,scilab training,MITCOE,"differential equation using scilab

[4] Bushan patel,scilab programming , runge kutta on second order

[5] www.x-engineer.org ,programming languages scilab"how to solve an ordinary equation in scilab"

[6] www.openeering.com ,numerical analysis using scilab ,jan 2015,modeling,IT00599320223

[7] Researchgate publication 23685145,"ordinary differential equation using euler,s technique and scilab programming

[8] 8. K.E Atkinson,"scilab textbook companion for an intrdectin to numerical analysis",john wiley and sons,ISBN:8126518502

[9] Kannan m.moudgalya,Scilab train- ing,MITCOE,"differential equation using Scilab ".

[10] Gilberto E Urroz,P.hd,P.E,"Ordinary differential equa- tion with Scilab",infoclearinghouse.com last accessed 2016/11/21.

[11] Scilab 6.0.1 in scilab help.