Naveen R

1RV17BT028

MATLAB SELF-STUDY ASSIGNMENT

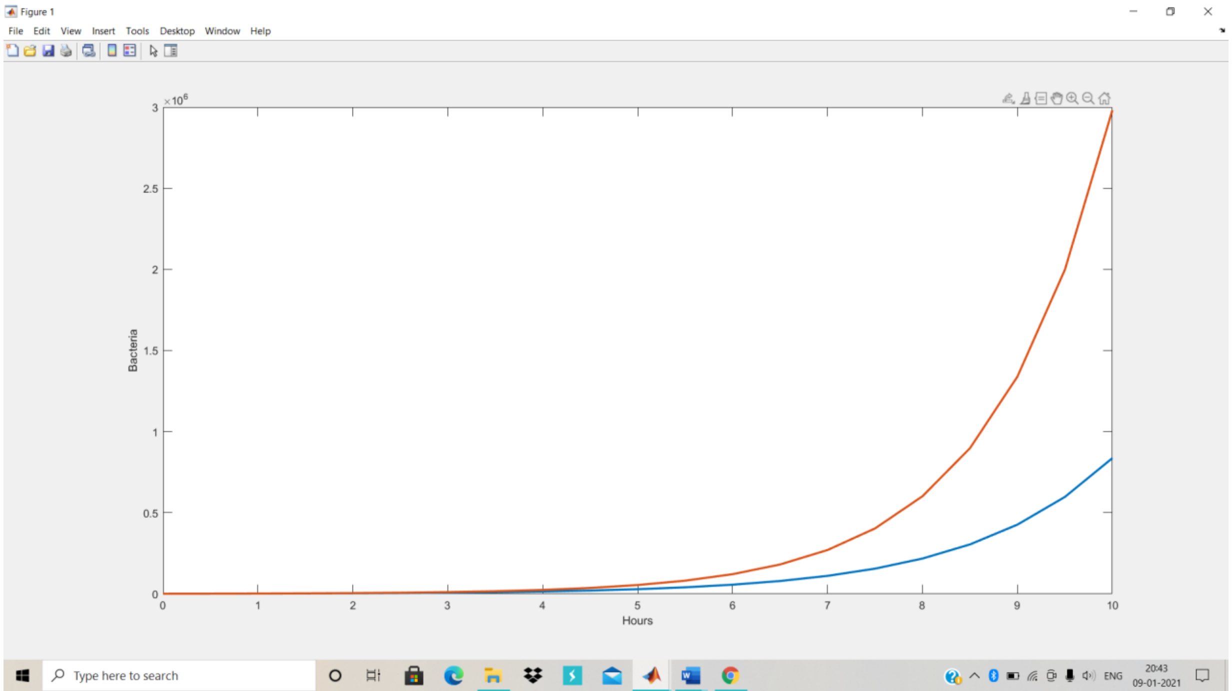
1) Illustrate Bacterial colony growth model with Differential Equation.

Specification: A colony of 1000 bacteria is multiplying at the rate of r = 0.8 per hour per individual

(i.e., an individual produces an average of 0.8 offspring every hour).

```
% applying Euler's algorithm
h = 0.5;
r = 0.8;
a = 0;
b = 10;
m = (b-a)/h;
N = zeros(1, m+1);
N(1) = 1000;
t = a:h:b;
for i = 1:m
    N(i+1) = N(i) +r*h*N(i);
end

Nex = N(1)*exp(r*t);
format bank
disp([t' N' Nex'])
plot(t,N,'linewidth',2), xlabel('Hours') ,
ylabel('Bacteria')
hold on
plot(t,Nex,'linewidth',2), hold off
```



2) Explain the following syntax involved in normal random numbers

a) r = normrnd(mu,sigma)

r = normrnd(mu,sigma) generates a random number from the normal distribution with mean parameter mu and standard deviation parameter sigma.b) r = normrnd(mu,sigma,sz1,...,szN)

r = normrnd(mu,sigma,sz1,...,szN) generates an array of normal random numbers, where sz1,...,szN indicates the size of each dimension.

c) r = normrnd(mu,sigma,sz)

r = normrnd(mu,sigma,sz) generates an array of normal random numbers, where vector sz specifies size(r).

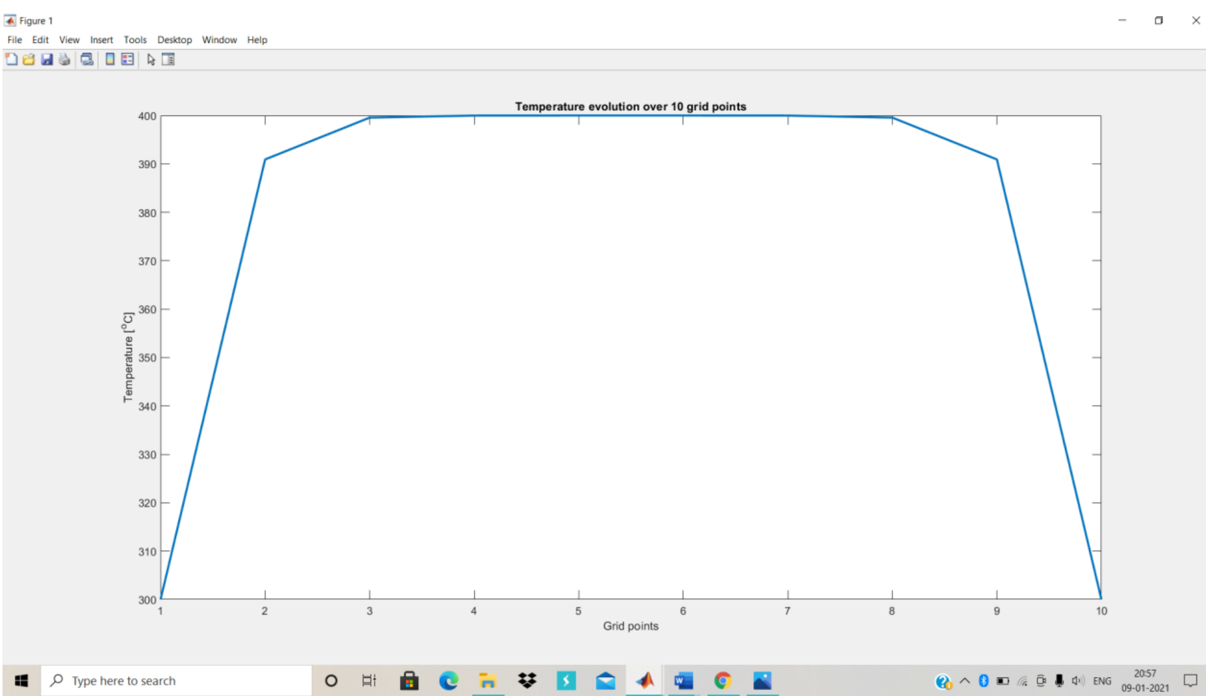
3) write a program to plot the temperature distribution in a insulated rod using the explicit Finite Central Difference Method and 1D Heat equation.

The rod is heated on one end at 400k and exposed to ambient temperature on the right end at 300k. I am using a time of 1s, 11 grid points and a .002s time step.

```
L=1;
t=1;
```

```
k=.001;
n=10;
nt=500;
dx=L/n;
dt=.002;
alpha=k*dt/dx^2;
T0=400*ones(1,n);
T1=300*ones(1,n);
T0(1) = 300;
T0(end) = 300;
for j=1:nt
   for i=2:n-1
       T1(i)=T0(i)+alpha*(T0(i+1)-2*T0(i)+T0(i-1));
   end
   T0=T1;
end
dlmwrite('Trial_1.csv',T1,'Delimiter','\n')

figure(1), clf
plot(T1,'linewidth',2);
xlabel('Grid points')
ylabel('Temperature [^oC]')
title(['Temperature evolution over 10 grid points'])
```



4) Explain the functions for Linear and Nonlinear Equations and Systems (any 5)

- equationsToMatrix   Convert linear equations to matrix form
- eliminate   Eliminate variables from rational equations
- finverse   Functional inverse
- linsolve   Solve linear equations in matrix form
- poles   Poles of expression or function
- `vpasolve`   Solve equations numerically

5) Mention any 2 functions for solving ODEs

- dsolve   Solve system of differential equations
- massMatrixForm   Extract mass matrix and right side of semilinear system of differential algebraic equations
- odeFunction   Convert symbolic expressions to function handle for ODE solvers
- odeToVectorField   Reduce order of differential equations to first-order

6) Mention the utility of decic and ode Function

Decic

- Find consistent initial conditions for first-order implicit ODE system with algebraic constraints
- [y0,yp0] = decic(eqs,vars,constraintEqs,t0,y0_est,fixedVars,yp0_est,options) finds consistent initial conditions for the system of first-order implicit ordinary differential equations with algebraic constraints returned by the reduceDAEToODE function.
- The call [eqs,constraintEqs] = reduceDAEToODE(DA_eqs,vars) reduces the system of differential algebraic equations DA_eqs to the system of implicit ODEs eqs. It also returns constraint equations encountered during system reduction. For the variables of

this ODE system and their derivatives, decic finds consistent initial conditions y0, yp0 at the time t0.

- Substituting the numerical values y0, yp0 into the differential equations subs(eqs, [t; vars(t); diff(vars(t))], [t0; y0; yp0]) and the constraint equations subs(constr, [t; vars(t); diff(vars(t))], [t0; y0; yp0]) produces zero vectors. Here, vars must be a column vector.
- y0_est specifies numerical estimates for the values of the variables vars at the time t0, and fixedVars indicates the values in y0_est that must not change during the numerical search. The optional argument yp0_est lets you specify numerical estimates for the values of the derivatives of the variables vars at the time t0.

ode Function

- Convert symbolic expressions to function handle for ODE solvers
- f = odeFunction(expr,vars) converts a system of symbolic algebraic expressions to a matlab function handle. This function handle can be used as input to the numerical matlab ode solvers, except for ode15i. The argument vars specifies the state variables of the system.
- f = odeFunction(expr,vars,p1,...,pN) specifies the symbolic parameters of the system as p1,...,pN.
- f = odeFunction(___,Name,Value) uses additional options specified by one or more Name,Value pair arguments.