Matt Pahayo

AE 5335 is taught by Dr. Riggins

Final Project

Propulsion 2

**1.0 Results**





Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

**2.0 Methodology**

First off, the problem is to solve a non-linear set of differential equations. The method that will be used to solve the non-linear set is a Newton-Raphson method for multivariable systems.

In this case or functions are the differential equations for continuity, momentum, energy, and the equation of state.

The Jacobian is needed for further calculation and is denoted [J].

(1)

At the first iteration {x} is the values at the inlet. To get the solution vector for equation 1, use an algorithm for solving linear equations. It was chosen that a Gauss elimination algorithm ought to be used. Alternatively, the built-in function in MATLAB may be used instead (linsolve uses LU factorization algorithm).

After solving equation 1, we can get the next value of {x}.

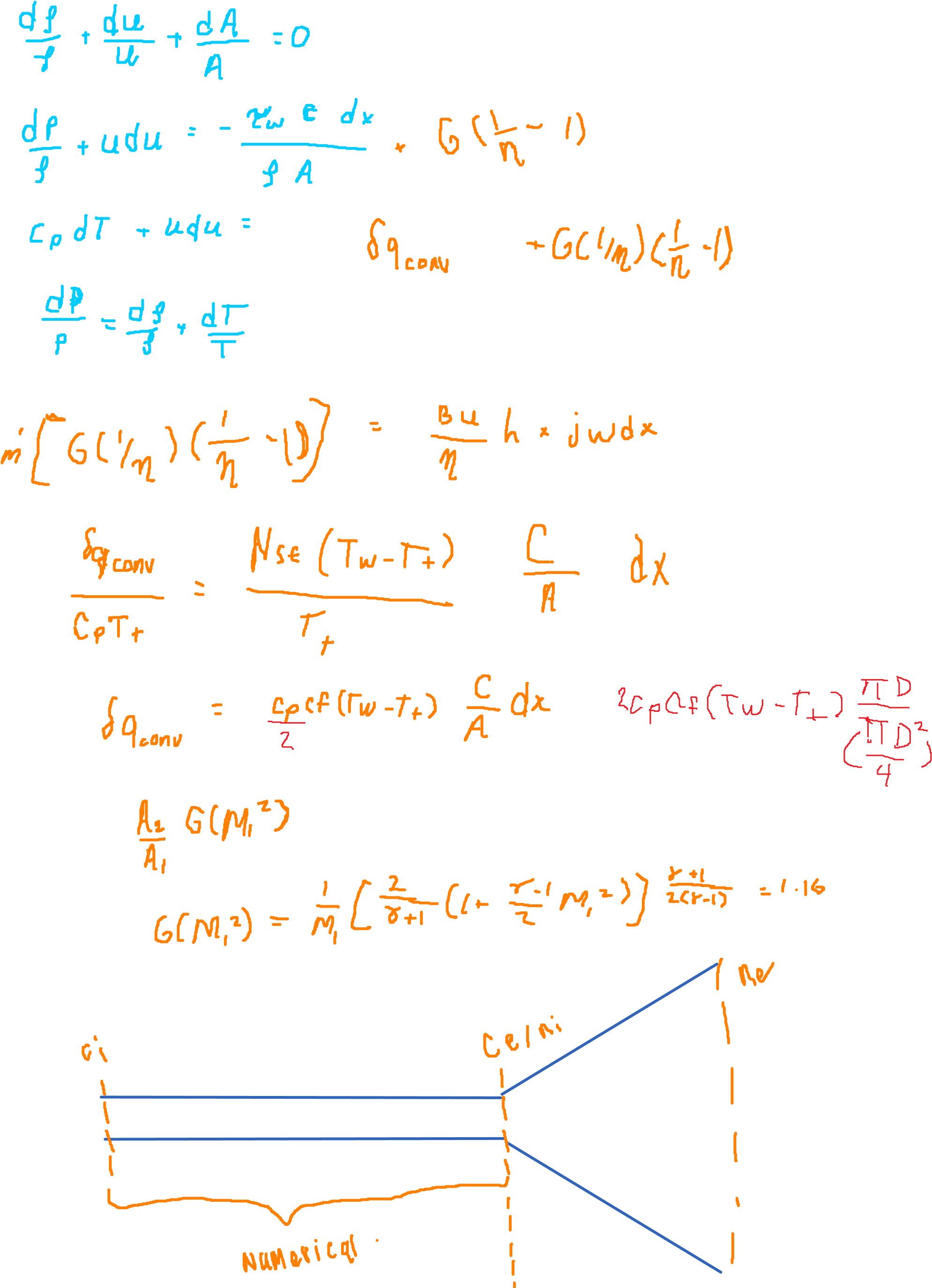
(2)

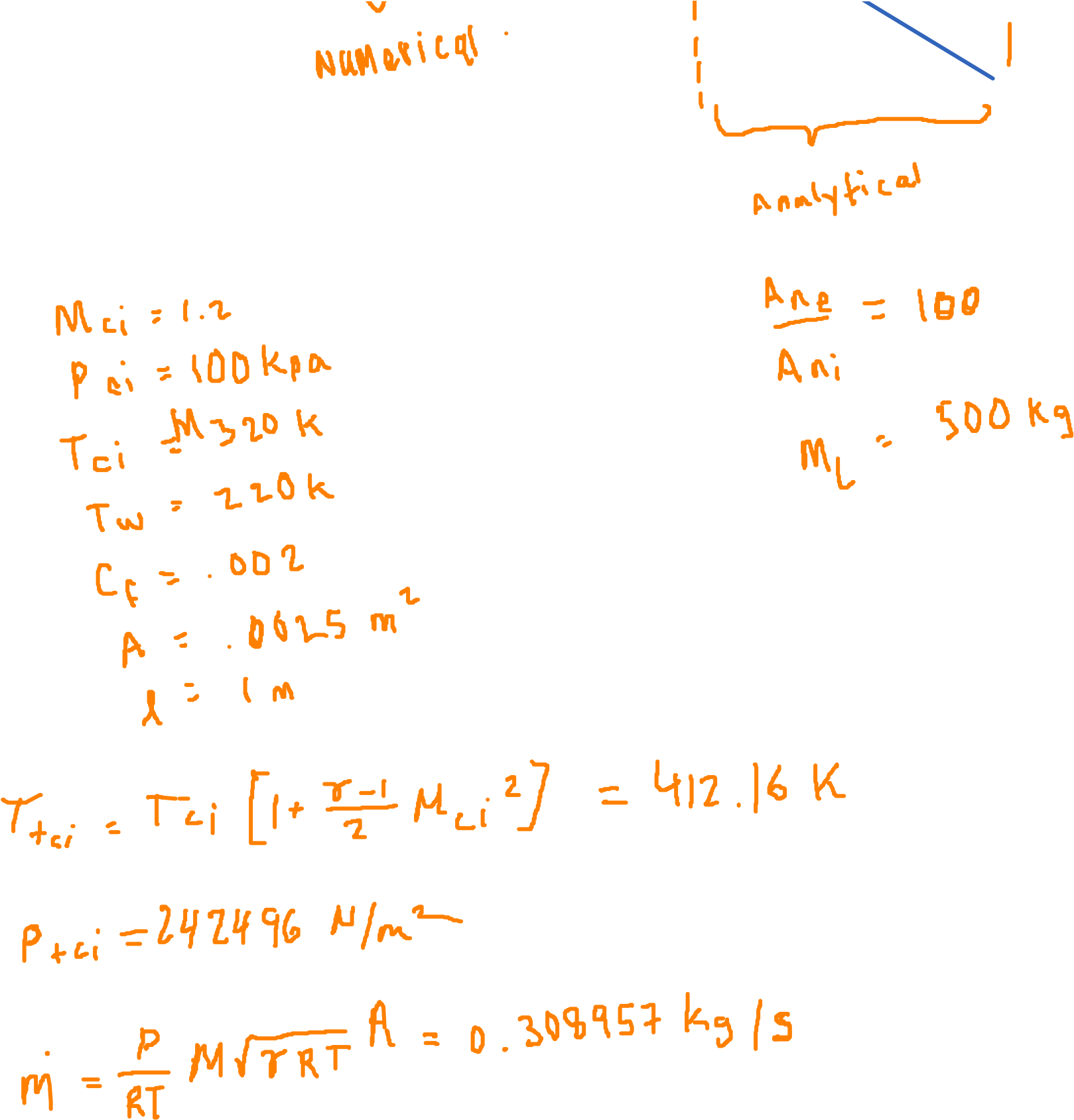
The exit criteria are the L2 norm of the current iteration of {f} scaled with the L2 norm of {f} at the original guess/inlet conditions. To exit this must be less than a tolerance value (ε). Epsilon was chosen as 10^-5. By decreasing epsilon more accurate results are found at the expense of computation time.

The amount that the steps are preformed are dependent on the step size used. Since a step size of 10000 was used, the number of times the Newton-Raphson algorithm is 10000. For each step, the {x} vector found at the end of the previous step is the initial guess of the next step.

To find the optimum values of B, sigma0, and eta to minimize the initial mass. A brute force method was used. There are 75 different combinations; at each combination of B, sigma0, and eta was put in to the MHD solver with 100 axial steps. For each combination initial mass was tabulated also total temperature was tabulated. The tabulated results will be given in the appendix. A minimum value of the initial mass was chosen with the constraint that the total temperature must not exceed 6000 K.

Sunday, April 25, 2021 9:35 PM





9388.87620582454

3.93814387583594

984.219759850445

4037.06812031099

53447.7580351589

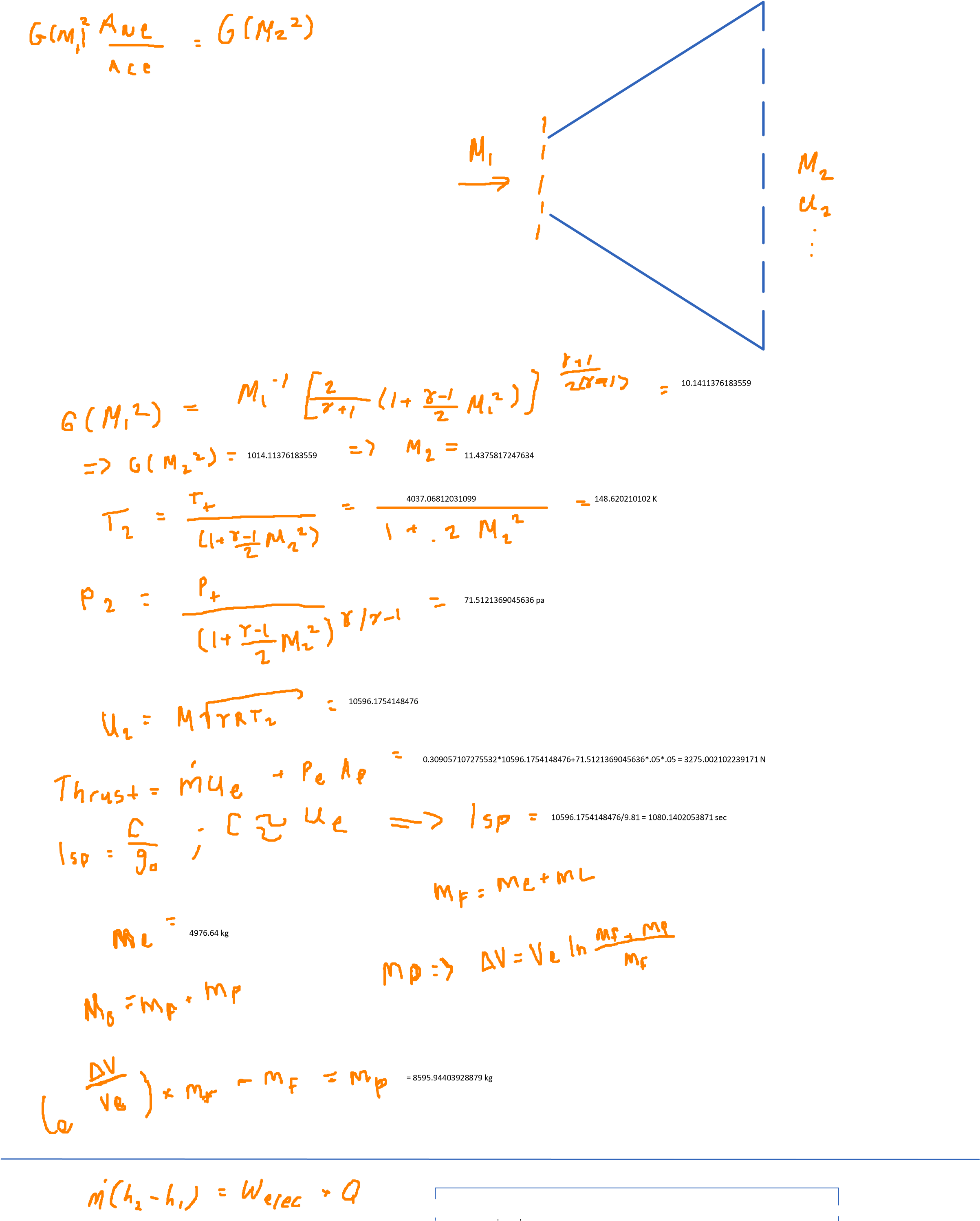
7470307.59327561

B = 1.4 T

Sig0 = 400 mho

Eta = 0.8

code





channel

=16169120.6650134 -16593873.561485 = --424752.896471599 W = -424.752

kW



16593873.561485 W = 16593.873561485 kW

0.309057107275532\*14437.5(4037.06812031099-412.16)= 16169120.6650134 W =16169.1206650134 kW



cooling

1. **Appendix**
   1. **IM.m**

% Iterative Methods class

% used to solve linear and non-linear systems iteratively

classdef IM

methods (Static)

%=========================================================================%

% Gauss-Seidel Method

%=========================================================================%

function [x,w] = gauSei(A,b,n,x,imax,es,lambda)

for i = 1:n

dum = A(i,i);

for j = 1:n

A(i,j) = A(i,j)/dum;

end

b(i) = b(i)/dum;

end

for i = 1:n

sum = b(i);

for j = 1:n

if i~= j

sum = sum - A(i,j)\*x(j);

end

x(i) = sum;

end

end

iter = 1;

sen = 0;

L2norm\_0 = norm(b-A\*x);

while sen == 0

sen = 1;

for i = 1:n

old = x(i);

sum = b(i);

for j = 1:n

if i~= j

sum = sum - A(i,j)\*x(j);

end

end

x(i) = lambda\*sum + (1-lambda)\*old;

L2norm = norm(b-A\*x);

if sen == 1 && x(i) ~= 0

ea = abs(L2norm/L2norm\_0)/1;

if ea > es

sen = 0;

end

end

end

iter = iter + 1;

if iter >= imax

break

end

end

w = [lambda iter];

end

%=========================================================================%

% Newton-Raphson Method

%=========================================================================%

function [q] = newRap(f,q,p,kmax)

% f is the 'A' matrix

% q is the 'b' vector

% p is the precision goal

% kmax is the maximum allowable iterations

syms x1 x2 x3 x4

fp(x1,x2,x3,x4) = jacobian(f,[x1 x2 x3 x4]);

b = transpose(double(f(q(1),q(2),q(3),q(4))));

b\_0 = b;

k = 0;

while (norm(b)/norm(b\_0)) > 10^p && k<kmax

A = double(fp(q(1),q(2),q(3),q(4)));

b = transpose(double(f(q(1),q(2),q(3),q(4))));

del = gauss(A,-b); % gauss elimination algorithm

q = q+del;

k = k + 1;

end

end

end

end

* 1. **Gauss elimination algorithm**

function [x] = gauss(a,b)

% gauss elimination

n = length(a);

k = 1 ;

p = k ;

big = abs(a(k,k));

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% pivoting portion

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

for ii=k+1:n

dummy = abs(a(ii,k));

if dummy > big

big = dummy;

p = ii ;

end

end

if p ~= k

for jj = k:n

dummy = a(p,jj);

a(p,jj) = a(k,jj);

a(k,jj) = dummy;

end

dummy = b(p);

b(p)=b(k);

b(k) = dummy;

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% elimination step

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

for k=1:(n-1)

for i=k+1:n

factor = a(i,k)/a(k,k);

for j=k+1:n

a(i,j) = a(i,j) - factor\*a(k,j);

end

b(i) = b(i) - factor\*b(k);

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% back substitution

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

x(n,1) = b(n)/a(n,n);

for i = n-1:-1:1

sum = b(i);

for j = i + 1:n

sum = sum - a(i,j)\*x(j,1);

end

x(i,1) = sum/a(i,i);

end

end

* 1. **main.m**

clc

clear all

close all

height = .05;

width = .05 ;

steps = 100;

cf = 0.002;

eta = .8;

Tw = 220;

w = 0;

h = 0;

A = ones(1,steps)\*height\*width;

% for convective heat tranmdot0sfer set ht to 1 else set it to 0

ht = 1;

l = 1;

sig0 = 400;

B = 1.4;

R = 4125;

gam =1.4;

M = 1.2

Pi = 100000

Ti = 320

[P,T,Pt2,Tt2,rho,u,M,thrust,F,mdot0,mdot,V,Ic,Powerc,forcec] = MHD(Pi,Ti,M,cf,Tw,eta,w,h,ht,A,l,sig0,B,steps,R,gam,height,width);

Gm1 = 1/M(end)\*(2/(gam+1)\*(1+(gam-1)/2\*M(end)^2))^((gam+1)/(2\*(gam-1)))

Gm2 = Gm1 \* 100

syms X

func = symfun(1/X\*(2/(gam+1)\*(1+(gam-1)/2\*X^2))^((gam+1)/(2\*(gam-1)))-Gm2,X)

Me = rootFind.newRap(func,7)

Te = Tt2(end)/(1+(gam-1)/2\*Me^2)

ue = Me\*sqrt(gam\*R\*Te)

ml = me(B,sig0,eta)

mp = exp(10000/ue)\*(ml+500)-(ml+500)

m0 = ml+500+ mp

Tt2(end)

function m\_e = me(B,sig0,eta)

N\_B = (B-.6)/.2;

N\_sig0 = (sig0-100)/100;

N\_eta = (1-eta)/.1;

m\_e = 1000\*1.2^(N\_B-1)\*1.2^(N\_sig0-1)\*2^(N\_eta-1);

end

**3.4**

% initialMass.m

clc

clear all

close all

y = 1

for i = 1:5

sig0 = i\*100+100;

for j = 1:5

B = j\*.2+.6;

for k = 1:3

eta = k\*.1+.6;

height = .05;

width = .05 ;

steps = 100;

cf = 0.002;

% eta = .9;

Tw = 220

w = 0;

h = 0;

A = ones(1,steps)\*height\*width;

% for convective heat transfer set ht to 1 else set it to 0

ht = 1;

l = 1;

% sig0 = 600;

% B = .8;

R = 4125;

gam =1.4;

M = 1.2;

Pi = 100000;

Ti = 320;

[P,T,Pt2,Tt2,rho,u,M,thrust,F,mdot0,mdot,V,Ic,Powerc,forcec] = MHD(Pi,Ti,M,cf,Tw,eta,w,h,ht,A,l,sig0,B,steps,R,gam,height,width);

Gm1 = 1/M(end)\*(2/(gam+1)\*(1+(gam-1)/2\*M(end)^2))^((gam+1)/(2\*(gam-1)))

Gm2 = Gm1 \* 100

syms X

func = symfun(1/X\*(2/(gam+1)\*(1+(gam-1)/2\*X^2))^((gam+1)/(2\*(gam-1)))-Gm2,X)

Me = rootFind.newRap(func,7)

Te = Tt2(end)/(1+(gam-1)/2\*Me^2)

ue = Me\*sqrt(gam\*R\*Te)

ml = me(B,sig0,eta)

mp = exp(10000/ue)\*(ml+500)-(ml+500)

m0(y,1) = ml+500+ mp

maxTt(y,1)=Tt2(end)

s(y,1)=sig0

b(y,1)=B

e(y,1)= eta

y = y+1

end

end

end

function m\_e = me(B,sig0,eta)

N\_B = (B-.6)/.2;

N\_sig0 = (sig0-100)/100;

N\_eta = (1-eta)/.1;

m\_e = 1000\*1.2^(N\_B-1)\*1.2^(N\_sig0-1)\*2^(N\_eta-1);

end

**3.5 MHD.m**

function [Pv,Tv,Pt2,Tt2,rho,uv,Mv,thrust,F,mdot0,mdot,V,Ic,Powerc,forcec] = MHD(P0,T0,M,cf,Tw,eta,w,h,ht,A,l,sig0,B,steps,R,gam,height,width)

% Author: Matt P

%{

MHD.m must include IM.m and gauss.m

P0 is the static pressure at the inlet

T0 is the static temperature at the inlet

M is the Mach # at the inlet

cf is the coefficient of skin friction

Tw is the wall temperature

eta is the thermodynamic efficiency

w is the work interaction per mass

h is the heat interaction per mass

ht is for convective heat transfer - 1 if present, 0 if not

A is the area along the length of the channel/nozzle and is a ROW VECTOR

l is the length of the device

%}

format longg

syms x1 x2 x3 x4

% x1 is P; x2 is rho; x3 is T; x4 is u

p = -7.0;

kmax = 1000;

T = T0;

P = P0;

cp = R\*gam/(gam-1);

rho = P/R/T;

rho0 = rho;

M0 = M;

u = M\*sqrt(gam\*R\*T);

u0 = u;

mdot = rho\*u\*A(1);

totalPower = 0;

Pv = zeros(steps-1,1);

rhov = zeros(steps-1,1);

Tv = zeros(steps-1,1);

uv = zeros(steps-1,1);

Mv = zeros(steps-1,1);

V = zeros(steps-1,1);

I = zeros(steps-1,1);

Power = zeros(steps-1,1);

F = zeros(steps-1,1);

Tt2 = zeros(steps-1,1);

Pt2 = zeros(steps-1,1);

Ic = zeros(steps-1,1);

Powerc = Ic;

forcec = Ic;

Pv(1,1)=P;

rhov(1,1)=rho;

Tv(1,1)=T;

uv(1,1)=u;

Mv(1,1)=M;

Pow = 0;

F = 0;

Icumulative = 0;

for i = 2:length(A)-1

G = A(i)\*sig0\*B^2\*u^2\*l/length(A)/mdot;

c = 2\*width+2\*height;

f = @(x1,x2,x3,x4) ([(x2-rho)/x2+(x4-u)/x4+(A(i+1)-A(i))/A(i) (x1-P)/x2+x4\*(x4-u)+1/2\*cf\*rho0\*u0^2\*c\*(l/length(A))/rho/A(i)-G\*(1/eta-1)-eta\*w/length(A) ...

cp\*(x3-T)+x4\*(x4-u)-ht\*1/2\*cp\*cf\*(Tw-T\*(1+(gam-1)/2\*M^2))\*c/A(i)\*(l/length(A))-G\*(1/eta-1)\*1/eta-(h/length(A))-(w/length(A)) (x3-T)/x3+(x2-rho)/x2-(x1-P)/x1]);

q = transpose([P,rho,T,u]);

[q] = IM.newRap(f,q,p,kmax);

P = q(1);

rho = q(2);

T = q(3);

u = q(4);

M = u/sqrt(gam\*R\*T);

Pv(i,1)=P;

rhov(i,1)=rho;

Tv(i,1)=T;

uv(i,1)=u;

Mv(i,1)=M;

F = Pv(i)/R/T\*uv(i,1)\*A(i)\*(uv(i,1)-uv(i-1,1))+Pv(i)\*A(i)-Pv(i-1)\*A(i-1) + F;

forcec(i-1,1) = F;

V(i-1,1) = B\*u/eta\*height;

I(i-1,1) = sig0\*(B\*u/eta-B\*u)\*width\*(l/length(A));

Icumulative = Icumulative + I(i-1,1);

Ic(i-1,1) = Icumulative;

Power(i,1) = V(i-1,1)\*I(i-1,1);

totalPower = totalPower + Power(i,1)

Powerc(i-1,1) = totalPower;

Pow = (G\*(1/eta-1)\*1/eta\*mdot + Pow)

Tt2(i,1) = T\*(1+(gam-1)/2\*M^2);

Pt2(i,1) = P\*(1+(gam-1)/2\*M^2)^(gam/(gam-1));

md = rho\*u\*A(i)

end

thrust = P/R/T\*u\*A(end)\*(u-u0)+P\*A(end)-P0\*A(1);

mdot0 = mdot;

mdot = rho\*u\*A(end);

if ht ==1

Qdot = mdot\*(cp\*(Tt2-T0\*(1+(gam-1)/2\*M0^2))-w);

end

**3.6 rootFind.m**

classdef rootFind

%rootFind is a class of functions that find the root of a function /

%data set

%------------------------------------------------------------------------%

methods (Static)

function x = Bisect(f,a,b,tol)

%Bisect uses the bisection algoritm using the interval

iter = 0;

while (b-a)/2 >= tol

c = (a+b)/2;

if f(c) > 0

b = c;

end

if f(c) < 0

a = c;

end

iter = iter + 1;

end

x = (a+b)/2

end

%------------------------------------------------------------------------%

function x = newRap(f,x0)

%newRap is a function that utilizes the Newton-Raphson

%algorithm to find the roots of the function

%x0 is the initial guess

fp = diff(f);

x=x0;

nmax=25;

eps=1;

n=0;

while eps>=1e-5&&n<=nmax

y=x-double(f(x))/double(fp(x));

eps=abs(y-x);

x=y;

n=n+1;

end

end

%------------------------------------------------------------------------%

end

end

**3.7 Tabulated data for finding m0**

****