B.Sc. Sixth Semester
DSC MATHEMATICS

DSC-2

Practicals on

NUMERICAL ANALYSIS

PRACTICAL MANUAL

[Using Maxima]

Prepared By

Lakkanna E N

Assistant Professor of Mathematics

Smt. Saraladevi Satheschandra Agarwal

Government First Grade College, Ballari-583101

CONTENTS

List of Programs

For Sixth Semester DSC 2 Mathematics

(Practicals on Numerical Analysis)

(4 Hours per Week and 56 hours per Semester)

1. Program to find root of an equation using Bisection and Regula-Falsi Methods.
2. Program to find root of an equation using Newton-Raphson and Secant Methods.
3. Program to solve system of algebraic equations using Gauss-Elimination Method.
4. Program to solve system of algebraic equations using Gauss-Jordan Method.
5. Program to solve system of algebraic equations using Gauss-Jacobi Method.
6. Program to solve system of algebraic equations using Gauss-Seidel Method.
7. Program solve system of algebraic equations using SOR Method.
8. Program to evaluate integral using Simpson's $1/3^{rd}$ and $3/8^{th}$ Rules.
9. Program to evaluate integral using Trapezoidal and Weddle Rules.
10. Program to find differentiation at specified point using Newton-Gregory interpolation method.
11. Program to find the missing value of table using Lagrange method.

Program 1

Program to find root of an equation using Bisection and Regula-Falsi Methods.

Aim: To find the approximate root of an algebraic / transcendental equation using Bisection and Regula-Falsi methods using Mathematics Softwares (FOSS).

Software: Maxima

Keys:

Key	Function
kill (all)	Unbinds all items on all infolists
float *(expr)*	Converts integers, rational numbers and bigfloats in *expr* to floating point numbers
numer:true	numer causes some mathematical functions (including exponentiation) with numerical arguments to be evaluated in floating point. Default value is false.
fpprintprec	This is an option variable to decide the number of digits to print when printing an ordinary float or bigfloat number. Default value is 16. Set any integer from 2 to 16.
:=	The function definition operator
define *(f(x_1, ..., x_n), expr)*	Defines a function named *f* with arguments *x_1*, …, *x_n* and function body *expr*.
[a_1, a_2,…,a_m]	List of numbers/objects a_1, a_2,…,a_m.
if cond_1 *then* expr_1 *else* expr_0	evaluates to expr_1 *if* cond_1 evaluates to true, otherwise the expression evaluates to expr_0.
print (“*text*”, *expr*)$	Displays *text* within inverted commas and evaluates and displays *expr*
block ([v_1, ..., v_m], expr_1, ..., expr_n)	The function *block* allows to make the variables *v_1*, …, *v_m* to be local for a sequence of commands.
go *(tag)*	*go* is used within a *block* to transfer control to the statement of the block which is tagged with the argument to go. To tag a statement, precede it by an atomic argument as another statement in the *block*. For example: *block* ([x], x:1, loop, x+1, ..., go(loop), ...)
push *(item, list)*	*push* prepends the item *item* to the list *list* and returns a copy of the new list
reverse *(list)*	Reverses the order of the members of the *list* (not the members themselves)
table_form()	Displays a 2D list in a form that is more readable than the output from *Maxima*’s default output routine. The input is a list of one or more lists.
log(x)	Represents the natural (base *e*) logarithm of *x*.
log(x)/log(10)	Represents the common (base 10) logarithm of *x*.
cos (x), sin(x), tan(x)	Trigonometric functions cosine, sine ant tangent of x respectively.
<=	less than or equal to
L[i]	Subscript operator for L_i

memoizing function *f*[*x_1, ..., x_n*] := *expr*	A memoizing function caches the result the first time it is called with a given argument, and returns the stored value, without recomputing it, when that same argument is given.

Definitions and Formulae:

Intermediate Value Theorem: Let $\boldsymbol{f(x)}$ be a real valued continuous function of the real variable $\boldsymbol{x}$. If $\boldsymbol{a}$ and $\boldsymbol{b}$ are two values such that $\boldsymbol{f(x)}$ has opposite signs (i.e. $\boldsymbol{f(a)\cdot f(b)<0}$) then there exists at least one real root of $\boldsymbol{f(x)=0}$ in the interval $(\boldsymbol{a,b})$.

Bisection Method or Interval Halving Method or Binary Chopping Method:

Let $\boldsymbol{f(x)}$ be continuous in $(\boldsymbol{a,b})$ and $\boldsymbol{f(a)\cdot f(b)<0}$. A real root of the equation $\boldsymbol{f(x)=0}$ lies in the interval $(\boldsymbol{a,b})$. In Bisection method, the first approximation to the root is given by $\boldsymbol{x_1=\frac{a+b}{2}}$, which is the bisecting point of the interval $(\boldsymbol{a,b})$. If $\boldsymbol{f(x_1)=0}$ then $\boldsymbol{x_1}$ is the required exact root. If $\boldsymbol{f(x_1)\neq 0}$ then the second approximation to the root is $\boldsymbol{x_2=\frac{a+x_1}{2}}$ if $\boldsymbol{f(a)\cdot f(x_1)<0}$ **or** $\boldsymbol{x_2=\frac{x_1+b}{2}}$ if $\boldsymbol{f(x_1)\cdot f(b)<0}$, which is the bisecting point of the respective interval $(\boldsymbol{a,x_1})$ **or** $(\boldsymbol{x_1,b})$. This iterative process is continued for the specified number of iterations or till root of desired accuracy is obtained.

Regula-Falsi Method or Method of False Position: Let $\boldsymbol{f(x)}$ is continuous in $(\boldsymbol{a,b})$ and $\boldsymbol{f(a)\cdot f(b)<0}$. A real root of the equation $\boldsymbol{f(x)=0}$ lies in the interval $(\boldsymbol{a,b})$. In Regula-Falsi method, the first approximation to the root $\boldsymbol{x_1}$ is the **x-intercept** of the chord joining $(\boldsymbol{a,f(a)})$ and $\left(\boldsymbol{b,f(b)}\right)$. A simple calculation gives the expression for $\boldsymbol{x_1}$ as

$$\boldsymbol{x_1=\frac{af(b)-bf(a)}{f(b)-f(a)}}$$

If $\boldsymbol{f(x_1)=0}$ then x_1 is the required exact root. If $\boldsymbol{f(a)\cdot f(x_1)<0}$ then the second approximation to the root $\boldsymbol{x_2}$ is the **x-intercept** of the chord joining $(\boldsymbol{a,f(a)})$ and $\left(\boldsymbol{x_1,f(x_1)}\right)$. The expression for $\boldsymbol{x_2}$ is

$$\boldsymbol{x_2=\frac{af(x_1)-x_1f(a)}{f(x_1)-f(a)}}$$

Otherwise, if $\boldsymbol{f(b)\cdot f(x_1)<0}$ then the second approximation to the root $\boldsymbol{x_2}$ is the **x-intercept** of the chord joining $(\boldsymbol{x_1,f(x_1)})$ and $\left(\boldsymbol{b,f(b)}\right)$. The expression for $\boldsymbol{x_2}$ is

$$\boldsymbol{x_2=\frac{x_1f(b)-bf(x_1)}{f(b)-f(x_1)}}$$

This iterative process is continued for the specified number of iterations or till root of desired accuracy is obtained.

Program: (Bisection Method)

Program to find the approximate root of given equation $f(x) = 0$ in the interval (a, b) using Bisection Method carrying n iterations.

```
kill(all)$
numer:true$
fpprintprec:7$
f(x):=given f(x)$
a: lower limit of the interval$
b: upper limit of the interval $
x[i]:=(a+b)/2$
n: no. of iterations$
print("Given equation is",f(x)=0)$
print("Given interval is (",a, ",",b,")")$
N:[["Iteration No.","Approximate root x", "f(x)"]]$
for i:1 thru n do (block(if f(a)*f(x[i])<0 then b:x[i]
        elseif f(b)*f(x[i])<0 then a:x[i]),N:push([i,x[i],f(x[i])],N))$
table_form(reverse(N))$
print("Approximate root by Bisection Method is x=",x[n])$
```

Program: (Bisection Method)

Program to find an approximate root of given equation $f(x) = 0$ in the interval (a, b) using Bisection Method up to given accuracy.

```
kill(all)$
numer:true$
fpprintprec:7$
f(x):= given f(x)$
a: lower limit of the interval $
b: upper limit of the interval $
i:1$
accuracy:0.00001$
x[i]:=(a+b)/2$
print("Given equation is",f(x)=0)$
print("Given interval is (",a, ",",b,")")$
N:[["Iteration No.","Approximate root x", "f(x)"]]$
block(loop,if f(a)*f(x[i])<0 then b:x[i] elseif f(b)*f(x[i])<0 then a:x[i],
   N:push([i,x[i],f(x[i])],N),if abs(x[i]-x[i-1])<=accuracy then
   (table_form(reverse(N)),print("Approximate root by Bisection Method is x=",x[i+1]))
else(i:i+1,go(loop)))$
```

Note: You may take **accuracy:0.001, 0.0001, 0.00001** to get approximate root correct to **2 decimal places, 3 decimal places, 4 decimal places** respectively.

Program: (Regula-Falsi Method)

Program to find the approximate root of given equation $f(x) = 0$ in the interval (a, b) using Regula-Falsi Method carrying n iterations.

```
kill(all)$
numer:true$
fpprintprec:7$
f(x):= given f(x)$
a: lower limit of the interval $
b: upper limit of the interval $
x[i]:=(a*f(b)-b*f(a))/(f(b)-f(a))$
n: no. of iterations$
print("Given equation is",f(x)=0)$
print("Given interval is (",a, ",",b,")")$
N:[["Iteration No.","Approximate root x", "f(x)"]]$
for i:1 thru n do (block(if f(a)*f(x[i])<0 then b:x[i]
      elseif f(b)*f(x[i])<0 then a:x[i]),N:push([i,x[i],f(x[i])],N))$
table_form(reverse(N))$
print("Approximate root by Regula-Falsi Method is x=",x[n])$
```

Program: (Regula-Falsi Method)

Program to find an approximate root of given equation $f(x) = 0$ in the interval (a, b) using Regula-Falsi Method up to given accuracy.

```
kill(all)$
numer:true$
fpprintprec:7$
f(x):= given f(x)$
a: lower limit of the interval $
b: upper limit of the interval $
i:1$
accuracy:0.00001$
x[i]:=(a*f(b)-b*f(a))/(f(b)-f(a))$
print("Given equation is",f(x)=0)$
print("Given interval is (",a, ",",b,")")$
N:[["Iteration No.","Approximate root x", "f(x)"]]$
block(loop,if f(a)*f(x[i])<0 then b:x[i] else a:x[i],N:push([i,x[i],f(x[i])],N),
   if abs(x[i]-x[i-1])<=accuracy then (table_form(reverse(N)),
      print("Approximate root by Regula-Falsi Method is x=",x[i]))
else(i:i+1,go(loop)))$
```

Note: You may take **accuracy:0.001, 0.0001, 0.00001** to get approximate root correct to **2 decimal places, 3 decimal places, 4 decimal places** respectively.

Worked Examples: Bisection Method

Problem 1. Write a program to find an approximate root of $x^3 - x - 1 = 0$ in the interval $(1,2)$ by Bisection Method. Carry out 15 iterations.

Program:

```
kill(all)$
numer:true$
fpprintprec:7$
f(x):=x^3-x-1$
a:1$
b:2$
x[i]:=float((a+b)/2)$
n:12$
print("Given equation is",f(x)=0)$
print("Given interval is (",a, ",",b,")")$
N:[["Iteration No.","Approximate root x", "f(x)"]]$
for i:1 thru n do (block(if f(a)*f(x[i])<0 then b:x[i]
     elseif f(b)*f(x[i])<0 then a:x[i]),N:push([i,x[i],f(x[i])],N))$
table_form(reverse(N))$
print("Approximate root by Bisection Method is x=",x[n])$
```

Output:

$Given\ equation\ is\ x^3 - x - 1 = 0$

$Given\ interval\ is\ (1,2)$

Iteration No.	Approximate root x	f(x)
1	1.5	0.875
2	1.25	−0.296875
3	1.375	0.2246094
4	1.3125	−0.05151367
5	1.34375	0.08261108
6	1.328125	0.01457596
7	1.320313	−0.01871061
8	1.324219	−0.002127945
9	1.326172	0.00620883
10	1.325195	0.002036651
11	1.324707	$-4.659488\ 10^{-5}$
12	1.324951	$9.94791\ 10^{-4}$
13	1.324829	$4.740388\ 10^{-4}$
14	1.324768	$2.137072\ 10^{-4}$
15	1.324738	$8.355244\ 10^{-5}$

$Approximate\ root\ by\ Bisection\ Method\ is\ x = 1.324738$

```
kill(all)$
numer:true$
fpprintprec:7$
f(x):=x^3−x−1$
a:1$
b:2$
x[i]:=float((a+b)/2)$
n:15$
print("Given equation is",f(x)=0)$
print("Given interval is (",a, ",",b,")")$
N:[["Iteration No.","Approximate root x", "f(x)"]]$
for i:1 thru n do (block(if f(a)·f(x[i])<0 then b:x[i]
     elseif f(b)·f(x[i])<0 then a:x[i]),N:push([i,x[i],f(x[i])],N))$
table_form(reverse(N))$
print("Approximate root by Bisection Method is x=",x[n])$
```

Given equation is $x^3 - x - 1 = 0$

Given interval is (1 , 2)

Iteration No.	Approximate root x	f(x)
1	1.5	0.875
2	1.25	−0.296875
3	1.375	0.2246094
4	1.3125	−0.05151367
5	1.34375	0.08261108
6	1.328125	0.01457596
7	1.320313	−0.01871061
8	1.324219	−0.002127945
9	1.326172	0.00620883
10	1.325195	0.002036651
11	1.324707	$-4.659488\ 10^{-5}$
12	1.324951	$9.94791\ 10^{-4}$
13	1.324829	$4.740388\ 10^{-4}$
14	1.324768	$2.137072\ 10^{-4}$
15	1.324738	$8.355244\ 10^{-5}$

Approximate root by Bisection Method is x= 1.324738

Problem 2. Write a program to find an approximate root of $x^3 - 2x - 5 = 0$ in the interval (2,3) by Bisection Method correct to 4 decimal places.

Program:

```
kill(all)$
numer:true$
fpprintprec:7$
f(x):=x^3-2·x-5$
a:2$
b:3$
i:1$
accuracy:0.00001$
x[i]:=(a+b)/2$
print("Given equation is",f(x)=0)$
print("Given interval is (",a, ",",b,")")$
N:[["Iteration No.","Approximate root x", "f(x)"]]$
block(loop,if f(a)·f(x[i])<0 then b:x[i] elseif f(b)·f(x[i])<0 then a:x[i],
   N:push([i,x[i],f(x[i])],N),if abs((x[i]-x[i-1]))<=accuracy then
   (table_form(reverse(N)),print("Approximate root by Bisection Method is x=",x[i+1]))
   else(i:i+1,go(loop)))$
```

Output:

Given equation is $x^3 - 2x - 5 = 0$

Given interval is (2 , 3 *)*

Iteration No.	*Approximate root x*	*f(x)*
1	2.5	5.625
2	2.25	1.890625
3	2.125	0.3457031
4	2.0625	−0.3513184
5	2.09375	−0.00894165
6	2.109375	0.1668358
7	2.101563	0.07856226
8	2.097656	0.03471428
9	2.095703	0.01286233
10	2.094727	0.001954348
11	2.094238	−0.003495149
12	2.094482	$-7.707752\ 10^{-4}$
13	2.094604	$5.916927\ 10^{-4}$
14	2.094543	$-8.956468\ 10^{-5}$
15	2.094574	$2.510581\ 10^{-4}$
16	2.094559	$8.074527\ 10^{-5}$
17	2.094551	$-4.410068\ 10^{-6}$

Approximate root by Bisection Method is x= 2.094555

Problem 3. Write a program to find an approximate root of $16x^3 - 95x^2 + 187x - 105 = 0$ in the interval (0,1) by Bisection Method correct to 4 decimal places.

Program:

```
kill(all)$
numer:true$
fpprintprec:7$
f(x):=16·x^3−95·x^2+187·x−105$
a:0$
b:1$
i:1$
accuracy:0.00001$
x[i]:=(a+b)/2$
print("Given equation is",f(x)=0)$
print("Given interval is (",a, ",",b,")")$
N:[["Iteration No.","Approximate root x", "f(x)"]]$
block(loop,if f(a)·f(x[i])<0 then b:x[i] elseif f(b)·f(x[i])<0 then a:x[i],
   N:push([i,x[i],f(x[i])],N),if abs((x[i+1]−x[i]))<=accuracy then
   (table_form(reverse(N)),print("Approximate root by Bisection Method is x=",x[i+1]))
   else(i:i+1,go(loop)))$
```

Output:

Given equation is $16\,x^3 - 95\,x^2 + 187\,x - 105 = 0$

Given interval is (0 , 1 *)*

Iteration No.	*Approximate root x*	*f(x)*
1	0.5	−33.25
2	0.75	−11.4375
3	0.875	−3.390625
4	0.9375	0.0

Approximate root by Bisection Method is x= 0.9375

Note: Observe little modification **<u>if abs(x[i+1]-x[i])<=accuracy</u>** in above problem as we got exact root.

Problem 4. Write a program to find an approximate root of $xe^x = 1$ in the interval $(0,1)$ by Bisection Method correct to 3 decimal places.

After rearrangement, given equation is $xe^x - 1 = 0 \Longrightarrow f(x) = xe^x - 1$

Program:

```
kill(all)$
numer:true$
fpprintprec:6$
f(x):=x·exp(x)−1$
a:0$
b:1$
i:1$
accuracy:0.0001$
x[i]:=(a+b)/2$
print("Given equation is",f(x)=0)$
print("Given interval is (",a, ",",b,")")$
N:[["Iteration No.","Approximate root x", "f(x)"]]$
block(loop,if f(a)·f(x[i])<0 then b:x[i] elseif f(b)·f(x[i])<0 then a:x[i],
  N:push([i,x[i],f(x[i])],N),if abs((x[i]−x[i−1]))<=accuracy then
  (table_form(reverse(N)),print("Approximate root by Bisection Method is x=",x[i+1]))
  else(i:i+1,go(loop)))$
```

Output:

Given equation is $x\ \%e^{x} - 1 = 0$

Given interval is (0 , 1 *)*

Iteration No.	Approximate root x	f(x)
1	0.5	−0.175639
2	0.75	0.58775
3	0.625	0.167654
4	0.5625	−0.0127818
5	0.59375	0.0751424
6	0.578125	0.0306192
7	0.570313	0.00878
8	0.566406	−0.00203538
9	0.568359	0.00336366
10	0.567383	$6.61983\ 10^{-4}$
11	0.566895	$-6.87237\ 10^{-4}$
12	0.567139	$-1.2762\ 10^{-5}$
13	0.567261	$3.24577\ 10^{-4}$
14	0.5672	$1.55899\ 10^{-4}$

Approximate root by Bisection Method is x= 0.567169

Worked Examples: Regula-Falsi Method

Problem 5. Write a program to find an approximate root of $2x - log_{10}x = 7$ in the interval (3.5,4) by Regula-Falsi Method. Carry out 5 iterations.

After rearrangement, given equation is $2x - log_{10}x - \mathbf{7} = \mathbf{0} \Longrightarrow \boldsymbol{f(x)} = 2x - \frac{\log(x)}{\log(10)} - \mathbf{7}$

Program:

```
kill(all)$
numer:true$
fpprintprec:7$
f(x):=2·x−log(x)/log(10)−7$
a:3.5$
b:4$
n:5$
x[i]:=(a·f(b)−b·f(a))/(f(b)−f(a))$
print("Given equation is",f(x)=0)$
print("Given interval is (",a, ",",b,")")$
N:[["Iteration No.","Approximate root x", "f(x)"]]$
for i:1 thru n do (block(if f(a)·f(x[i])<0 then b:x[i]
      elseif f(b)·f(x[i])<0 then a:x[i]),N:push([i,x[i],f(x[i])],N))$
table_form(reverse(N))$
print("Approximate root by Regula-Falsi Method is x=",x[n])$
```

Output:

Given equation is $-0.4342945\log(x)+2x-7=0$

Given interval is (3.5 , 4 *)*

Iteration No.	*Approximate root x*	*f(x)*
1	3.788781	$-9.375136\ 10^{-4}$
2	3.789277	$-1.525885\ 10^{-6}$
3	3.789278	$-2.483274\ 10^{-9}$
4	3.789278	$-4.041212\ 10^{-12}$
5	3.789278	$-6.217249\ 10^{-15}$

Approximate root by Regula−Falsi Method is x= 3.789278

Problem 6. Write a program to find an approximate root of $x^{2.2} = 69$ in the interval $(5,8)$ by Regula-Falsi Method correct to 4 decimal places.

After rearrangement, given equation is $x^{2.2} - 69 = \mathbf{0} \Longrightarrow \boldsymbol{f(x)} = x^{2.2} - 69$

Program:

```
kill(all)$
numer:true$
fpprintprec:7$
f(x):=x^2.2−69$
a:5$
b:8$
i:1$
accuracy:0.00001$
x[i]:=(a·f(b)−b·f(a))/(f(b)−f(a))$
print("Given equation is",f(x)=0)$
print("Given interval is (",a, ",",b,")")$
N:[["Iteration No.","Approximate root x", "f(x)"]]$
block(loop,if f(a)·f(x[i])<0 then b:x[i] else a:x[i],N:push([i,x[i],f(x[i])],N),
   if abs(x[i]−x[i−1])<=accuracy then (table_form(reverse(N)),
      print("Approximate root by Regula-Falsi Method is x=",x[i]))
else(i:i+1,go(loop)))$
```

Output:

Given equation is $x^{2.2} - 69 = 0$

Given interval is (5 , 8)

Iteration No.	*Approximate root x*	*f(x)*
1	6.65599	-4.275625
2	6.834002	-0.4061477
3	6.85067	-0.03755399
4	6.852209	-0.003463682
5	6.852351	$-3.193886\ 10^{-4}$
6	6.852364	$-2.945041\ 10^{-5}$
7	6.852365	$-2.71558\ 10^{-6}$

Approximate root by Regula−Falsi Method is x= 6.852365

Exercise:

Write a program to find an approximate root of the given equations in the given interval by Bisection Method.

1. $x^3 + 5x - 11 = 0$ in (1,2). Carry out 15 iterations (Answer: $x = 1.51059$)

2. $\sin(x) + x^2 - 1 = 0$ in (0,1) correct to 4 decimal places (Answer: $x = 0.6367264$)

3. $x\, log_{10}\, x - 1.2 = 0$ in (2,3) correct to 4 decimal places (Answer: $x = 2.740643$)

4. $2x = \cos(x) + 3$ in (1,2) correct to 4 decimal places (Answer: $x = 1.523594$)

5. $x^x = 100$ in (3,4) correct to 4 decimal places (Answer: $x = 3.597286$)

Write a program to find an approximate root of the given equations in the given interval by Regula-Falsi Method.

1. $x^3 - 3x + 4 = 0$ in $(-3, -2)$. Carry out 11 iterations (Answer: $x = -2.195822$)

2. $e^x - x - 2 = 0$ in (1,1.4) correct to 4 decimal places (Answer: $x = 1.146193$)

3. $\cos(x) - 1.3x = 0$ in (0,1) correct to 4 decimal places (Answer: $x = 0.6241845$)

4. $xe^x = 1$ in (0,1) correct to 4 decimal places (Answer: $x = 0.5671418$)

5. $x^3 - 2x - 5 = 0$ in (2,3) correct to 4 decimal places (Answer: $x = 2.094547$)

Program 2

Program to find root of an equation using Newton-Raphson and Secant Methods.

Aim: To find the approximate root of an equation using Newton-Raphson and Secant methods using Mathematics Softwares (FOSS).

Software: Maxima

Keys:

Key	Function
kill (all)	Unbinds all items on all infolists
float *(expr)*	Converts integers, rational numbers and bigfloats in *expr* to floating point numbers
numer:true	numer causes some mathematical functions (including exponentiation) with numerical arguments to be evaluated in floating point. Default value is false.
fpprintprec	This is an option variable to decide the number of digits to print when printing an ordinary float or bigfloat number. Default value is 16. Set any integer from 2 to 16.
:=	The function definition operator
define *(f(x_1, …, x_n), expr)*	Defines a function named *f* with arguments *x_1*, …, *x_n* and function body *expr*.
memoizing function *f*[*x_1*, ..., *x_n*] := *expr*	A memoizing function caches the result the first time it is called with a given argument, and returns the stored value, without recomputing it, when that same argument is given.
[a_1, a_2,…,a_m]	List of numbers/objects a_1, a_2,…,a_m.
if cond_1 *then* expr_1 *else* expr_0	evaluates to expr_1 *if* cond_1 evaluates to true, otherwise the expression evaluates to expr_0.
print (“*text*”, *expr*)$	Displays *text* within inverted commas and evaluates and displays *expr*
block ([v_1, …, v_m], expr_1, …, expr_n)	The function *block* allows to make the variables *v_1*, …, *v_m* to be local for a sequence of commands.
push *(item, list)*	*push* prepends the item *item* to the list *list* and returns a copy of the new list
reverse *(list)*	Reverses the order of the members of the *list* (not the members themselves)
table_form()	Displays a 2D list in a form that is more readable than the output from *Maxima*’s default output routine. The input is a list of one or more lists.
<=	less than or equal to
L[i]	Subscript operator for L_i
diff (expr, x)	Returns the first derivative of expr with respect to the variable x
exp(x) or %e^x	e^x , exponential function.

Note:1. Press Shift+Enter for evaluation of commands and display of output.
2. Replace semicolon (;) by dollar ($) to suppress output of any input line and vice-versa.
3. Start each session with kill(all)$ or quit()$ to remove previously assigned values of all symbols

Definitions and Formulae:

Newton-Raphson Method: The Newton-Raphson method also known as Newton's method, is an iterative numerical method used to find the roots of a real-valued function. This formula is named after Sir Isaac Newton and Joseph Raphson, as they independently contributed to its development. In this method, part of the curve between the initial guess and x-axis is replaced by means of the tangent to the curve at that point. The Newton-Raphson Method has a convergence of order 2 which means it has a quadratic convergence. Derivation of Newton's iteration formula is as follows: Let $f(x) = 0$ be the given equation and $x = x_0$ be the initial approximation to the exact root. If h is the error such that $x = x_0 + h$ is the exact root then $f(x_0 + h) = 0$. Using Taylor's expansion of $f(x_0 + h)$ we have,

$$f(x_0) + hf'(x_0) + \frac{h^2}{2!}f''(x_0) + \frac{h^3}{3!}f'''(x_0) + \cdots = 0$$

Since h is very small, neglecting terms containing $h^2, h^3, \ldots,$ we get

$$f(x_0) + hf'(x_0) = 0 \Rightarrow h = -\frac{f(x_0)}{f'(x_0)}$$

Thus, the first approximation to the root is given by $x_1 = x_0 + h,$ i.e.,

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

provided $f'(x_0) \neq 0$. The second approximation is

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

and so on. In general Newton-Raphson iteration formula is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Secant Method: The secant method is a root-finding procedure in numerical analysis that uses a series of roots of secant lines to better approximate a root of a function $f(x)$. The convergence is particularly superlinear, but not really quadratic ($\approx$ 1.168). Iteration formula for secant method is derived as follows: Let $f(x) = 0$ be the given equation and $x = x_0$ and $x = x_1$ be two initial limits to the exact root. Then the next approximation x_2 to the root is taken as the **x-intercept** of the secant line joining the initial points $(x_0, f(x_0))$ and $(x_1, f(x_1))$. A simple calculation gives a formula for x_2 as

$$x_2 = x_0 - \frac{f(x_0)(x_1 - x_0)}{f(x_1) - f(x_0)} = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}$$

In general, successive approximations by Secant method are given by the iteration formula,

$$x_{n+1} = \frac{x_{n-1} f(x_n) - x_n f(x_{n-1})}{f(x_n) - f(x_{n-1})}$$

Program: (Newton-Raphson Method)

Program to find the approximate root of given equation $f(x) = 0$ near x_0 using Newton-Raphson Method carrying n iterations.

```
kill(all)$
fpprintprec:7$
define(f(x), LHS of given equation)$
define(g(x),x-f(x)/diff(f(x),x))$
x[0]:given initial root x0$
x[i]:=float(g(x[i-1]))$
n:given number of iterations$
print(“Given equation is”,f(x)=0)$
print(“Initial Approximation is “,’x[0]=x[0])$
N:[[“Iteration No.”,”Approximate root x”,”f(x)”]]$
for i:1 thru n do N:push([i,x[i],f(x[i])],N)$
table_form(reverse(N))$
print(“Approximate root by Newton’s Method is x=”,x[n])$
```

Note: If interval (a, b) containing a root is given, one can take $x_0 = a$ or $x_0 = b$ or $x_0 = \frac{a+b}{2}$.

Program: (Newton-Raphson Method)

Program to find the approximate root of given equation $f(x) = 0$ near x_0 using Newton-Raphson Method up to given accuracy.

```
kill(all)$
fpprintprec:7$
define(f(x), LHS of given equation)$
define(g(x),x-f(x)/diff(f(x),x))$
x[0]:given initial root x0$
x[i]:=float(g(x[i-1]))$
i:1$
accuracy:0.00001$
print(“Given equation is”,f(x)=0)$
print(“Initial Approximation is “,’x[0]=x[0])$
N:[[“Iteration No.”,”Approximate root x”,”f(x)”]]$
block(loop,N:push([i,x[i],f(x[i])],N),
   if abs(x[i]-x[i-1])<=accuracy then (table_form(reverse(N)),
      print(“Approximate root by Newton’s Method is x=”,x[i]))
else(i:i+1,go(loop)))$
```

Note: 1. If interval (a, b) containing a root is given, one can take $x_0 = a$ or $x_0 = b$ or $x_0 = \frac{a+b}{2}$.

2. You may take **accuracy:0.001, 0.0001, 0.00001** to get approximate root correct to **2 decimal places, 3 decimal places, 4 decimal places** respectively.

Program: (Secant Method)

Program to find the approximate root of given equation $f(x) = 0$ from the initial guess x_0 and x_1 using Secant Method carrying n iterations.

```
kill(all)$
fpprintprec:7$
f(x):=LHS of given equation$
x[0]:given initial guess x0$
x[1]: given initial guess x1$
x[i]:=float((x[i-2]*f(x[i-1])-x[i-1]*f(x[i-2]))/(f(x[i-1])-f(x[i-2])))$
n:given number of iterations$
print("Given equation is",f(x)=0)$
print("Initial Approximations are ",'x[0]=x[0],"and",'x[1]=x[1])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
for i:2 thru n do (N:push([i-1,x[i],f(x[i])],N))$
table_form(reverse(N))$
print("Approximate root by Secant Method is x=",x[n])$
```

Program: (Secant Method)

Program to find the approximate root of given equation $f(x) = 0$ from the initial guess x_0 and x_1 using Secant Method up to given accuracy.

```
kill(all)$
fpprintprec:7$
f(x):=LHS of given equation$
x[0]:given initial guess x0$
x[1]: given initial guess x1$
x[i]:=float((x[i-2]*f(x[i-1])-x[i-1]*f(x[i-2]))/(f(x[i-1])-f(x[i-2])))$
i:2$
accuracy:0.00001$
print("Given equation is",f(x)=0)$
print("Initial Approximations are ",'x[0]=x[0],"and",'x[1]=x[1])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
block(loop,N:push([i-1,x[i],f(x[i])],N),
   if abs(x[i]-x[i-1])<=accuracy then (table_form(reverse(N)),
      print("Approximate root by Secant Method is x=",x[i]))
else(i:i+1,go(loop)))$
```

Note: You may take **accuracy:0.001, 0.0001, 0.00001** to get approximate root correct to **2 decimal places, 3 decimal places, 4 decimal places** respectively.

Worked Examples: Newton's Method

Problem 1. Write a program to find an approximate root of $x \cdot sin(x) + cos(x) = 0$ near $x_0 = \pi$ by Newton's Method. Carry out **4** iterations

Program:

```
kill(all)$
fpprintprec:7$
define(f(x),x*sin(x)+cos(x))$
define(g(x),x-f(x)/diff(f(x),x))$
x[0]:π$
x[i]:=float(g(x[i-1]))$
n:4$
print("Given equation is",f(x)=0)$
print("Initial Approximation is ",'x[0]=x[0])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
for i:1 thru n do N:push([i,x[i],f(x[i])],N)$
table_form(reverse(N))$
print("Approximate root by Newton's Method is x=",x[n])$
```

Output:

$Given\ equation\ is\ x\sin(x) + \cos(x) = 0$

$Initial\ Approximation\ is\ x_0 = \pi$

$Iteration\ No.$	$Approximate\ root\ x$	$f(x)$
1	2.823283	-0.06618607
2	2.7986	-5.63571510^{-4}
3	2.798386	-4.30508810^{-8}
4	2.798386	-1.11022310^{-16}

$Approximate\ root\ by\ Newton's\ Method\ is\ x = 2.798386$

Problem 2. Write a program to find an approximate value of fourth root of **22** i.e., $\boldsymbol{x = \sqrt[4]{22}}$ taking $\boldsymbol{x_0 = 2}$ by Newton's Method upto accuracy of 0.00001.

After rearrangement, given equation is $x^4 - 22 = \mathbf{0} \Longrightarrow \boldsymbol{f(x)} = x^4 - 22$

Program:

```
kill(all)$
fpprintprec:7$
define(f(x),x^4-22)$
define(g(x),x-f(x)/diff(f(x),x))$
x[0]:2$
x[i]:=float(g(x[i-1]))$
i:1$
accuracy:0.00001$
print("Given equation is",f(x)=0)$
print("Initial Approximation is ",'x[0]=x[0])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
block(loop,N:push([i,x[i],f(x[i])],N),
   if abs(x[i]-x[i-1])<=accuracy then (table_form(reverse(N)),
      print("Approximate root by Newton's Method is x=",x[i]))
else(i:i+1,go(loop)))$
```

Output:

$Given\ equation\ is\ x^4 - 22 = 0$

$Initial\ Approximation\ is\ x_0 = 2$

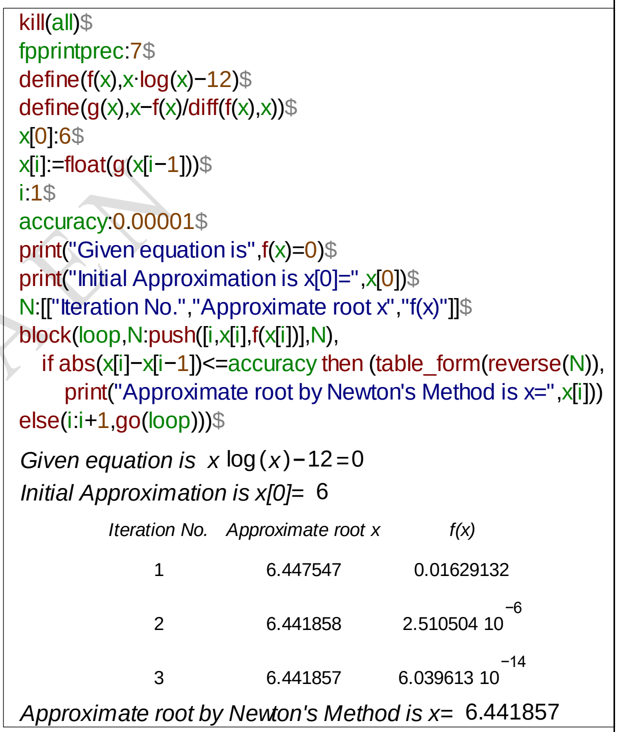
$Iteration\ No.$	$Approximate\ root\ x$	$f(x)$
1	2.1875	0.8977203
2	2.166059	0.01311237
3	2.165737	$2.928659 10^{-6}$
4	2.165737	$1.49214 10^{-13}$

$Approximate\ root\ by\ Newton's\ Method\ is\ x = 2.165737$

Problem 3. Write a program to find an approximate root of $\boldsymbol{x \cdot log(x) - 12 = 0}$ near $\boldsymbol{x_0 = 6}$ by Newton's Method correct to 4 decimal places.

Program:

```
kill(all)$
fpprintprec:7$
define(f(x),x*log(x)-12)$
define(g(x),x-f(x)/diff(f(x),x))$
x[0]:6$
x[i]:=float(g(x[i-1]))$
i:1$
accuracy:0.00001$
print("Given equation is",f(x)=0)$
print("Initial Approximation is ",'x[0]=x[0])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
block(loop,N:push([i,x[i],f(x[i])],N),
   if abs(x[i]-x[i-1])<=accuracy then (table_form(reverse(N)),
      print("Approximate root by Newton's Method is x=",x[i]))
else(i:i+1,go(loop)))$
```



```
kill(all)$
fpprintprec:7$
define(f(x),x·log(x)−12)$
define(g(x),x−f(x)/diff(f(x),x))$
x[0]:6$
x[i]:=float(g(x[i−1]))$
i:1$
accuracy:0.00001$
print("Given equation is",f(x)=0)$
print("Initial Approximation is x[0]=",x[0])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
block(loop,N:push([i,x[i],f(x[i])],N),
   if abs(x[i]−x[i−1])<=accuracy then (table_form(reverse(N)),
      print("Approximate root by Newton's Method is x=",x[i]))
else(i:i+1,go(loop)))$
```

Given equation is $x\log(x) - 12 = 0$

Initial Approximation is x[0]= 6

Iteration No.	*Approximate root x*	*f(x)*
1	6.447547	0.01629132
2	6.441858	$2.510504\ 10^{-6}$
3	6.441857	$6.039613\ 10^{-14}$

Approximate root by Newton's Method is x= 6.441857

Output:

$Given\ equation\ is\ x\ log(x) - 12 = 0$

$Initial\ Approximation\ is\ x_0 = 6$

$Iteration\ No.$	$Approximate\ root\ x$	$f(x)$
1	6.447547	0.01629132
2	6.441858	$2.510504 10^{-6}$
3	6.441857	$6.039613 10^{-14}$

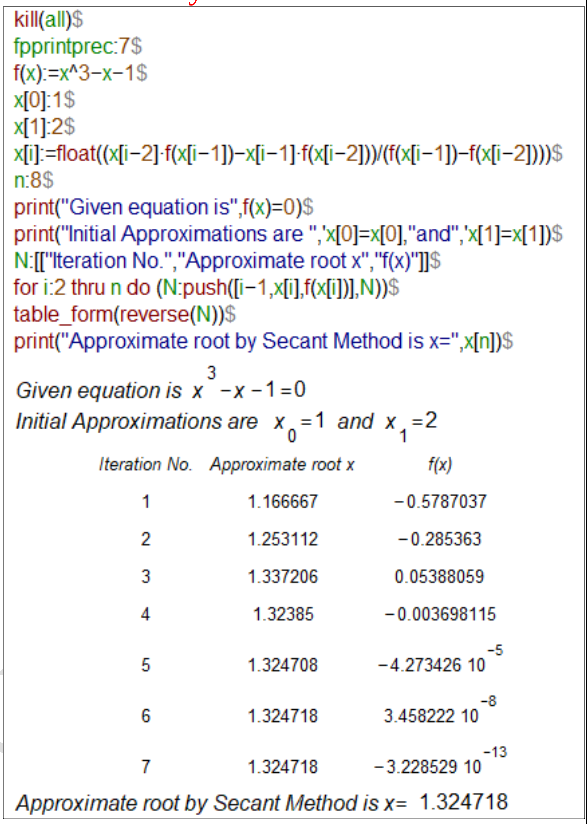
$Approximate\ root\ by\ Newton's\ Method\ is\ x = 6.441857$

Worked Examples: Secant Method

Problem 1. Write a program to find an approximate root of $x^3 - x - 1 = 0$ using initial approximations $x_0 = 1$ and $x_1 = 2$ by Secant Method. Carry out 7 iterations.

Program:

```
kill(all)$
fpprintprec:7$
f(x):=x^3-x-1$
x[0]:1$
x[1]:2$
x[i]:=float((x[i-2]*f(x[i-1])-x[i-1]*f(x[i-2]))/(f(x[i-1])-f(x[i-2])))$
n:8$
print("Given equation is",f(x)=0)$
print("Initial Approximations are ",'x[0]=x[0],"and",'x[1]=x[1])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
for i:2 thru n do (N:push([i-1,x[i],f(x[i])],N))$
table_form(reverse(N))$
print("Approximate root by Secant Method is x=",x[n])$
```



```
kill(all)$
fpprintprec:7$
f(x):=x^3−x−1$
x[0]:1$
x[1]:2$
x[i]:=float((x[i−2]·f(x[i−1])−x[i−1]·f(x[i−2]))/(f(x[i−1])−f(x[i−2])))$
n:8$
print("Given equation is",f(x)=0)$
print("Initial Approximations are ",'x[0]=x[0],"and",'x[1]=x[1])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
for i:2 thru n do (N:push([i−1,x[i],f(x[i])],N))$
table_form(reverse(N))$
print("Approximate root by Secant Method is x=",x[n])$
```

Given equation is $x^3 - x - 1 = 0$

Initial Approximations are $x_0 = 1$ *and* $x_1 = 2$

Iteration No.	Approximate root x	f(x)
1	1.166667	−0.5787037
2	1.253112	−0.285363
3	1.337206	0.05388059
4	1.32385	−0.003698115
5	1.324708	$-4.273426\ 10^{-5}$
6	1.324718	$3.458222\ 10^{-8}$
7	1.324718	$-3.228529\ 10^{-13}$

Approximate root by Secant Method is x= 1.324718

Output:

$$Given\ equation\ is\ x^3 - x - 1 = 0$$

$$Initial\ Approximations\ are\ \ x_0 = 1\ \ and\ \ x_1 = 2$$

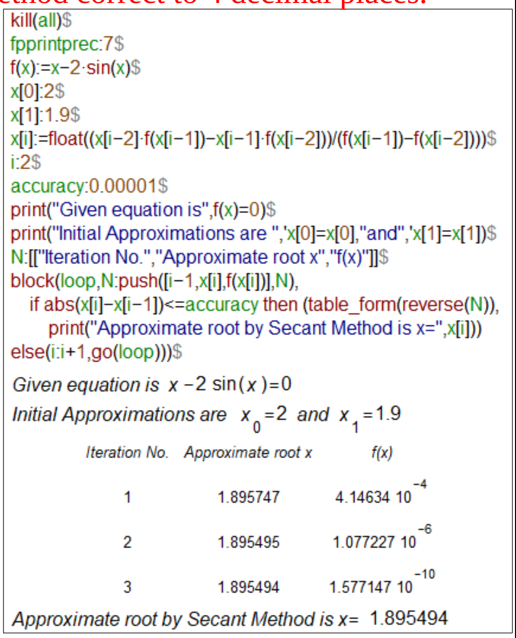
$Iteration\ No.$	$Approximate\ root\ x$	$f(x)$
1	1.166667	-0.5787037
2	1.253112	-0.285363
3	1.337206	0.05388059
4	1.32385	-0.003698115
5	1.324708	-4.27342610^{-5}
6	1.324718	3.45822210^{-8}
7	1.324718	-3.22852910^{-13}

$$Approximate\ root\ by\ Secant\ Method\ is\ x = 1.324718$$

Problem 2. Write a program to find an approximate root of $x - 2\sin(x) = 0$ using initial approximations $x_0 = 2$ and $x_1 = 1.9$ by Secant Method correct to 4 decimal places.

Program:

```
kill(all)$
fpprintprec:7$
f(x):=x-2*sin(x)$
x[0]:2$
x[1]:1.9$
x[i]:=float((x[i-2]*f(x[i-1])-x[i-1]*f(x[i-2]))/(f(x[i-1])-f(x[i-2])))$
i:2$
accuracy:0.00001$
print("Given equation is",f(x)=0)$
print("Initial Approximations are ",'x[0]=x[0],"and",'x[1]=x[1])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
block(loop,N:push([i-1,x[i],f(x[i])],N),
   if abs(x[i]-x[i-1])<=accuracy then (table_form(reverse(N)),
```



```
kill(all)$
fpprintprec:7$
f(x):=x−2·sin(x)$
x[0]:2$
x[1]:1.9$
x[i]:=float((x[i−2]·f(x[i−1])−x[i−1]·f(x[i−2]))/(f(x[i−1])−f(x[i−2])))$
i:2$
accuracy:0.00001$
print("Given equation is",f(x)=0)$
print("Initial Approximations are ",'x[0]=x[0],"and",'x[1]=x[1])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
block(loop,N:push([i−1,x[i],f(x[i])],N),
  if abs(x[i]−x[i−1])<=accuracy then (table_form(reverse(N)),
    print("Approximate root by Secant Method is x=",x[i]))
else(i:i+1,go(loop)))$
```

Given equation is $x - 2\sin(x) = 0$

Initial Approximations are $x_0 = 2$ *and* $x_1 = 1.9$

Iteration No.	Approximate root x	f(x)
1	1.895747	$4.14634\ 10^{-4}$
2	1.895495	$1.077227\ 10^{-6}$
3	1.895494	$1.577147\ 10^{-10}$

Approximate root by Secant Method is x= 1.895494

```
        print("Approximate root by Secant Method is x=",x[i]))
  else(i:i+1,go(loop)))$
```

Output:

$Given\ equation\ is\ x - 2\sin(x) = 0$

$Initial\ Approximations\ are\ \ x_0 = 2\ \ and\ \ x_1 = 1.9$

$Iteration\ No.$	$Approximate\ root\ x$	$f(x)$
1	1.895747	4.1463410^{-4}
2	1.895495	1.07722710^{-6}
3	1.895494	1.57714710^{-10}

$Approximate\ root\ by\ Secant\ Method\ is\ x = 1.895494$

Problem 3. Write a program to find an approximate time x for which the processes governed by $f_1(x) = 100(1 - e^{-0.2x})$ and $f_2(x) = 40e^{-0.01x}$ reach the same temperature using initial approximations $x_0 = 2$ and $x_1 = 3$ by Secant Method correct to 4 decimal places.

For same temperature, $f_1(x) = f_2(x) \Longrightarrow$, $f_1(x) - f_2(x) = 0$

$$\Longrightarrow \boldsymbol{f(x) = 100(1 - e^{-0.2x}) - 40e^{-0.01x}}$$

Program:

```
kill(all)$
fpprintprec:7$
f(x):=100*(1-exp(-0.2*x))-40*exp(-0.01*x)$
x[0]:2$
x[1]:3$
x[i]:=float((x[i-2]*f(x[i-1])-x[i-1]*f(x[i-2]))/(f(x[i-1])-f(x[i-2])))$
i:2$
accuracy:0.00001$
print("Given equation is",f(x)=0)$
print("Initial Approximations are ",'x[0]=x[0],"and",'x[1]=x[1])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
block(loop,N:push([i-1,x[i],f(x[i])],N),
   if abs(x[i]-x[i-1])<=accuracy then (table_form(reverse(N)),
      print("Approximate root by Secant Method is x=",x[i]))
else(i:i+1,go(loop)))$
```

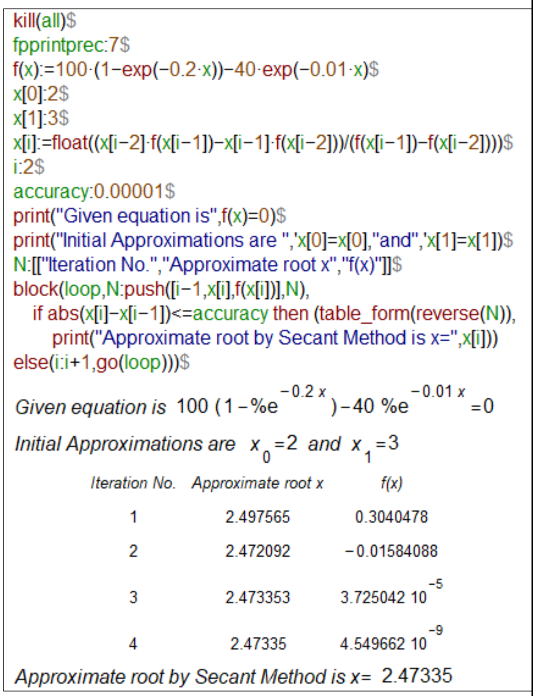
Output:

$Given\ equation\ is\ 100(1 - \%e^{-0.2x}) - 40\%e^{-0.01x} = 0$

$Initial\ Approximations\ are\ \ x_0 = 2\ \ and\ \ x_1 = 3$

$Iteration\ No.$	$Approximate\ root\ x$	$f(x)$
1	2.497565	0.3040478
2	2.472092	−0.01584088
3	2.473353	3.72504210^{-5}
4	2.47335	4.54966210^{-9}

$Approximate\ root\ by\ Secant\ Method\ is\ x = 2.47335$



```
kill(all)$
fpprintprec:7$
f(x):=100·(1−exp(−0.2·x))−40·exp(−0.01·x)$
x[0]:2$
x[1]:3$
x[i]:=float((x[i−2]·f(x[i−1])−x[i−1]·f(x[i−2]))/(f(x[i−1])−f(x[i−2])))$
i:2$
accuracy:0.00001$
print("Given equation is",f(x)=0)$
print("Initial Approximations are ",'x[0]=x[0],"and",'x[1]=x[1])$
N:[["Iteration No.","Approximate root x","f(x)"]]$
block(loop,N:push([i−1,x[i],f(x[i])],N),
   if abs(x[i]−x[i−1])<=accuracy then (table_form(reverse(N)),
      print("Approximate root by Secant Method is x=",x[i]))
else(i:i+1,go(loop)))$
```

$Given\ equation\ is\ 100\,(1-\%e^{-0.2\,x})-40\,\%e^{-0.01\,x}=0$

$Initial\ Approximations\ are\ \ x_0=2\ \ and\ \ x_1=3$

Iteration No.	Approximate root x	f(x)
1	2.497565	0.3040478
2	2.472092	−0.01584088
3	2.473353	$3.725042\ 10^{-5}$
4	2.47335	$4.549662\ 10^{-9}$

Approximate root by Secant Method is x= 2.47335

Exercise:

Write a program to find an approximate root of the given equations near given initial value by Newton-Raphson Method.

1. $x^3 - 2x - 5 = 0$ given $x_0 = 2$. Carry out 4 iterations (Answer: $x = 2.094551$)

2. $x^3 + 5x - 11 = 0$ given $x_0 = 1$. Carry out 4 iterations (Answer: $x = 1.510595$)

3. $xe^x - 2 = 0$ given $x_0 = 1$ correct to 4 decimal places (Answer: $x = 0.8526055$)

4. $xe^x - \cos(x) = 0$ given $x_0 = 0.5$ correct to 4 decimal places (Answer: $x = 0.5177574$)

5. $tan(x) - x = 0$ given $x_0 = 4.5$ correct to 4 decimal places (Answer: $x = 4.493409$)

6. Find cube root of 15 given $x_0 = 2.5$ correct to 4 decimal places (Answer: $x = 2.466212$)

7. $3x - cos(x) - 1 = 0$ given $x_0 = 0.6$ correct to 4 decimal places (Answer: $x = 0.6071016$)

Write a program to find an approximate root of the given equations from given initial values by Secant Method.

1. $x^3 - 2x - 5 = 0$ given $x_0 = 2$ and $x_1 = 3$. Carry out 6 iterations(n=7)

   (Answer: $x = 2.094551$)

2. $x^3 - 5x - 6 = 0$ given $x_0 = 2$ and $x_1 = 3$ correct to 4 decimal places.

   (Answer: $x = 2.689095$)

3. $1 - \frac{x^2}{4} + \frac{x^4}{64} - \frac{x^6}{2304} = 0$ given $x_0 = 2$ and $x_1 = 2.5$ correct to 4 decimal places.

   (Answer: $x = 2.391647$)

4. $e^{-x} - \tan(x) = 0$ given $x_0 = 1$ and $x_1 = 0.7$ correct to 4 decimal places.

   (Answer: $x = 0.5313909$)

5. $\cos(x)\cosh(x) - 1 = 0$ given $x_0 = 4$ and $x_1 = 5$ correct to 4 decimal places.

   (Answer: $x = 4.730041$)

Program 3

Program to solve system of algebraic equations using Gauss-Elimination Method.

Aim: To check consistency of a system of linear algebraic equations and to solve if consistent by Gauss-Elimination Method using Mathematics Softwares (FOSS).

Software: Maxima

Keys:

Key	**Function**
kill (all)	Unbinds all items on all infolists
linsolve_params	When linsolve_params is true, linsolve also generates the %r symbols used to represent arbitrary parameters. Otherwise, linsolve solves an under-determined system of equations with some variables expressed in terms of others.
linsolve ([expr_1, …, expr_m], [x_1, …, x_n])	Solves the list of simultaneous linear equations for the list of variables. The expressions must each be polynomials in the variables and may be equations.
length (expr)	Returns (by default) the number of parts in the external (displayed) form of *expr*. For lists this is the number of elements.
pop (list)	pop removes and returns the first element from the list list.
coefmatrix ([eqn_1, …, eqn_m], [x_1, …, x_n])	Returns the coefficient matrix for the variables x_1, …, x_n of the system of linear equations eqn_1, …, eqn_m.
augcoefmatrix ([eqn_1, …, eqn_m], [x_1, …, x_n])	Returns the augmented coefficient matrix for the variables x_1, …, x_n of the system of linear equations eqn_1, …, eqn_m.
triangularize (M)	Returns the upper triangular form of the matrix M, as produced by Gaussian elimination
list_matrix_entries (M)	Returns a list containing the elements of the matrix M
matrix_size *(M)*	Return a two-member list that gives the number of rows and columns, respectively of the matrix M.
and	The logical conjunction operator.
if cond_1 *then* expr_1 *else* expr_0	evaluates to expr_1 *if* cond_1 evaluates to true, otherwise the expression evaluates to expr_0.
print ("*text*", *expr*)$	Displays *text* within inverted commas and evaluates and displays *expr*

Note:1. Press Shift+Enter for evaluation of commands and display of output.
2. Replace semicolon (;) by dollar ($) to suppress output of any input line and vice-versa.
3. Start each session with kill(all)$ or quit()$ to remove previously assigned values of all symbols

Definitions and Formulae:

Linear Algebraic Equation: An equation $\boldsymbol{f(x_1, x_2, x_3 \dots, x_n) = 0}$ is called a linear algebraic equation in variables $\boldsymbol{x_1, x_2, x_3 \dots, x_n}$ if $\boldsymbol{f(x_1, x_2, x_3 \dots, x_n)}$ is a polynomial with rational coefficients in which all variables $\boldsymbol{x_1, x_2, x_3 \dots, x_n}$ appear in first degree only. For example, $2x + 3 = 0$, $x - 3y = 0$, $-\frac{1}{2x} + 3y - \frac{z}{3} = \frac{11}{2}$, $2t - 5x + 3y - 4z = 14$, are all linear algebraic equations in one, two, three, four variables respectively. The most general form of linear algebraic equation in n variables is $\boldsymbol{a_1x_1 + a_2x_2 + a_3x_3 + \cdots + a_nx_n = b}$ where $\boldsymbol{b, a_i \in Q(the\ set\ of\ rational\ numbers)}\ \forall\ \boldsymbol{i}$ and $\boldsymbol{a_i \neq 0}$ for some $\boldsymbol{i = 1, 2, 3, .. n}$

System of Linear Algebraic Equations: Two or more linear algebraic equations taken together (simultaneously) is called a system of linear algebraic equations. Most general form of a system of $\boldsymbol{n}$ linear algebraic equations in $\boldsymbol{m}$ variables is:

$$\begin{array}{c}
\boldsymbol{a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1m}x_m = b_1} \\
\boldsymbol{a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2m}x_m = b_2} \\
\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\
\boldsymbol{a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nm}x_m = b_n}
\end{array}$$

In matrix form, this system of Linear algebraic equations is $\boldsymbol{AX = B}$ where,

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{bmatrix}$$ and is called the Coefficient Matrix,

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x_1} \\ \boldsymbol{x_2} \\ \vdots \\ \boldsymbol{x_m} \end{bmatrix}$$ and is called the matrix of variables (Variable Matrix) and

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \vdots \\ \boldsymbol{b_n} \end{bmatrix}$$ and is called the matrix of constants (Constant Matrix) and

$$[\boldsymbol{A}|\boldsymbol{B}] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} & b_1 \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} & b_m \end{bmatrix}$$ and is called the augmented matrix.

Consistent System: A system of linear equations $\boldsymbol{AX = B}$ is said to be consistent if it has a solution. Consistent system may have unique solution (Only one solution) or may have infinitely many solutions. If $\boldsymbol{AX = B}$ has no solution, then it is called inconsistent system.

Solving a system of linear algebraic equations: The process of finding a solution of a system of linear equations $\boldsymbol{AX = B}$ is called solving a system of linear equations. There are two methods of solving a system of linear equations: Direct methods and Iterative methods. Direct methods give exact solution of a consistent system in finite steps. Gauss-Elimination, Gauss-Jordan, Matrix Inversion, Cramer's rule, LU decomposition are examples of direct methods. On the other hand, iterative methods are methods in which an initial approximation to solution is used to generate sequence of solutions which may converge to the exact solution. Gauss-Jacobi, Gauss- Seidel method and SOR methods are examples of iterative methods.

Gauss-Elimination Method: It is one of the direct methods of solving a system of linear algebraic equations $\boldsymbol{AX = B}$. In this method, the augmented matrix $[\boldsymbol{A}|\boldsymbol{B}]$ is reduced to echelon form $[\boldsymbol{U}|\boldsymbol{C}]$ and then the reduced system $\boldsymbol{UX = C}$ is solved by back-substitution.

Program:

Program to test consistency and hence to find solution of given system of linear algebraic equations $AX = B$ using Gauss-Elimination Method.

```
kill(all)$
linsolve_params:false$
EQ:[given equations separated by comma]$
X:[variables separated by comma]$
S:linsolve(EQ,X)$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print(" Required solution is",S)
else print("Given system has infinite solutions") and
print(" Required solution is",S)$
```

Program:

Program to find coefficient matrix, augmented matrix, reduced/upper triangular system of given system of linear algebraic equations $AX = B$ and hence to find solution using Gauss-Elimination Method.

```
kill(all)$
linsolve_params:false$
EQ:[given equations separated by comma]$
X:[variables separated by comma]$
S:linsolve(EQ,X)$
A:augcoefmatrix (EQ, X)$
B:coefmatrix (EQ, X)$
C:col (triangularize(A), matrix_size(A)[2])$
D:triangularize(B).X$
E:map("=",list_matrix_entries(D),list_matrix_entries(-C))$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Augmented Matrix is", A)$
print("Reduced Augmented Matrix is", triangularize(A))$
print("Reduced System of Equations is")$
while E # [ ] do disp(popI)$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print(" Required solution is",S)
else print("Given system has infinite solutions") and
print(" Required solution is",S)$
```

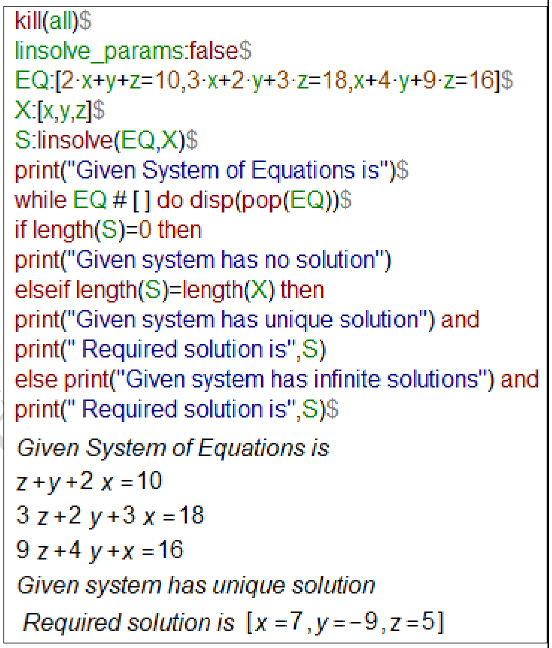
Worked Examples:

Problem 1. Write a program to test consistency and find solution if consistent of given system of equations by Gaussian elimination:

$2x + y + z = 10, 3x + 2y + 3z = 18, x + 4y + 9z = 16$

Program:

```
kill(all)$
linsolve_params:false$
EQ:[2*x+y+z=10,3*x+2*y+3*z=18,x+4*y+9*z=16]$
X:[x,y,z]$
S:linsolve(EQ,X)$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print(" Required solution is",S)
else print("Given system has infinite solutions") and
print(" Required solution is",S)$
```



```
kill(all)$
linsolve_params:false$
EQ:[2·x+y+z=10,3·x+2·y+3·z=18,x+4·y+9·z=16]$
X:[x,y,z]$
S:linsolve(EQ,X)$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print(" Required solution is",S)
else print("Given system has infinite solutions") and
print(" Required solution is",S)$
```

Given System of Equations is

$z+y+2x=10$

$3z+2y+3x=18$

$9z+4y+x=16$

Given system has unique solution

Required solution is $[x=7, y=-9, z=5]$

Output:

$Given\ System\ of\ Equations\ is$

$z + y + 2x = 10$

$3z + 2y + 3x = 18$

$9z + 4y + x = 16$

$Given\ system\ has\ unique\ solution$

$Required\ solution\ is\ [x = 7, y = -9, z = 5]$

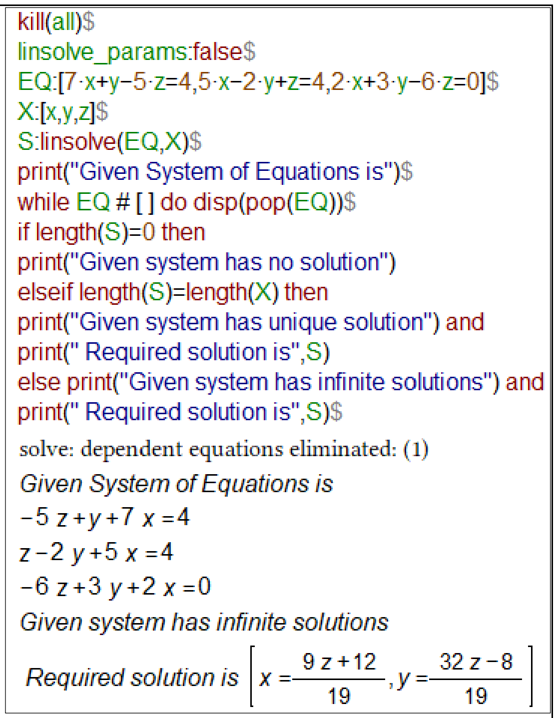
Problem 2. Write a program to test consistency and find solution if consistent of given system of equations by Gaussian elimination:

$7x + y - 5z = 4, 5x - 2y + z = 4, 2x + 3y - 6z = 0$

Program:

```
kill(all)$
linsolve_params:false$
EQ:[7*x+y-5*z=4,5*x-2*y+z=4,2*x+3*y-6*z=0]$
X:[x,y,z]$
S:linsolve(EQ,X)$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print(" Required solution is",S)
```

```
else print("Given system has infinite solutions") and
print(" Required solution is",S)$
```



```
kill(all)$
linsolve_params:false$
EQ:[7·x+y−5·z=4,5·x−2·y+z=4,2·x+3·y−6·z=0]$
X:[x,y,z]$
S:linsolve(EQ,X)$
print("Given System of Equations is")$
while EQ # [] do disp(pop(EQ))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print(" Required solution is",S)
else print("Given system has infinite solutions") and
print(" Required solution is",S)$
solve: dependent equations eliminated: (1)
```

Given System of Equations is

$-5z+y+7x=4$

$z-2y+5x=4$

$-6z+3y+2x=0$

Given system has infinite solutions

Required solution is $\left[x=\frac{9z+12}{19}, y=\frac{32z-8}{19}\right]$

Output:

$$Given\ System\ of\ Equations\ is$$

$$-5z+y+7x=4$$

$$z-2y+5x=4$$

$$-6z+3y+2x=0$$

$$Given\ system\ has\ infinite\ solutions$$

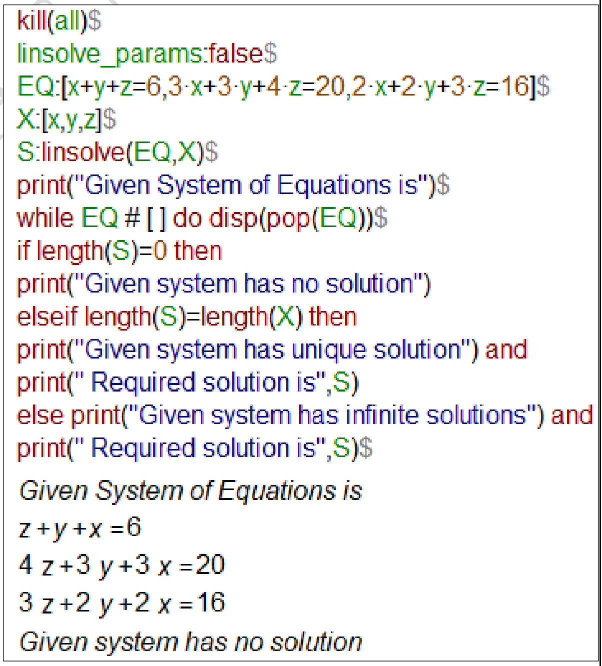
$$Required\ solution\ is\ \left[x=\frac{9z+12}{19}, y=\frac{32z-8}{19}\right]$$

Problem 3. Write a program to test consistency and find solution if consistent of given system of equations by Gaussian elimination:

$x+y+z=6, x+3y+4z=20, 2x+2y+3z=16$

Program:

```
kill(all)$
linsolve_params:false$
EQ:[x+y+z=6,3*x+3*y+4*z=20,2*x+2*y+3*z=16]$
X:[x,y,z]$
S:linsolve(EQ,X)$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print(" Required solution is",S)
else print("Given system has infinite solutions") and
print(" Required solution is",S)$
```



```
kill(all)$
linsolve_params:false$
EQ:[x+y+z=6,3·x+3·y+4·z=20,2·x+2·y+3·z=16]$
X:[x,y,z]$
S:linsolve(EQ,X)$
print("Given System of Equations is")$
while EQ # [] do disp(pop(EQ))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print(" Required solution is",S)
else print("Given system has infinite solutions") and
print(" Required solution is",S)$
```

Given System of Equations is

$z+y+x=6$

$4z+3y+3x=20$

$3z+2y+2x=16$

Given system has no solution

Output:

$$Given\ System\ of\ Equations\ is$$

$$z+y+x=6$$

$$4z+3y+3x=20$$

$$3z+2y+2x=16$$

$$Given\ system\ has\ no\ solution$$

Problem 4. Write a program to find augmented matrix, reduced system and test consistency and find solution if consistent of given system of equations by Gaussian elimination:

$$5x_1 + x_2 + x_3 + x_4 = 4,$$
$$x_1 + 7x_2 + x_3 + x_4 = 12,$$
$$x_1 + x_2 + 6x_3 + x_4 = -5,$$
$$x_1 + x_2 + x_3 + 4x_4 = -6$$

Program:

```
kill(all)$
linsolve_params:false$
EQ:[5·x[1]+x[2]+x[3]+x[4]=4,x[1]+7·x[2]+x[3]+x[4]=12,x[1]+x[2]+6·x[3]+x[4]=−5,x[1]+x[2]+x[3]+4·x[4]=−6]$
X:[x[1],x[2],x[3],x[4]]$
S:linsolve(EQ,X)$
A:augcoefmatrix (EQ, X)$
B:coefmatrix (EQ, X)$
C:col (triangularize(A), matrix_size(A)[2])$
D:triangularize(B).X$
E:map("=",list_matrix_entries(D),list_matrix_entries(−C))$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Augmented Matrix is", A)$
print("Reduced Augmented Matrix is", triangularize(A))$
print("Reduced System of Equations is")$
while E # [ ] do disp(pop(E))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print(" Required solution is",S)
else print("Given system has infinite solutions") and
print(" Required solution is",S)$
```

Output:

Given System of Equations is

$x_4+x_3+x_2+5\,x_1=4$

$x_4+x_3+7\,x_2+x_1=12$

$x_4+6\,x_3+x_2+x_1=-5$

$4\,x_4+x_3+x_2+x_1=-6$

Augmented Matrix is $\begin{pmatrix} 5 & 1 & 1 & 1 & -4 \\ 1 & 7 & 1 & 1 & -12 \\ 1 & 1 & 6 & 1 & 5 \\ 1 & 1 & 1 & 4 & 6 \end{pmatrix}$

Reduced Augmented Matrix is $\begin{pmatrix} 5 & 1 & 1 & 1 & -4 \\ 0 & 34 & 4 & 4 & -56 \\ 0 & 0 & 194 & 24 & 242 \\ 0 & 0 & 0 & 702 & 1404 \end{pmatrix}$

Reduced System of Equations is

$x_4+x_3+x_2+5\,x_1=4$

$4\,x_4+4\,x_3+34\,x_2=56$

$24\,x_4+194\,x_3=-242$

$702\,x_4=-1404$

Given system has unique solution

Required solution is $[x_1=1,x_2=2,x_3=-1,x_4=-2]$

Exercise:

Write a program to test consistency and to find solution if consistent of given system of equations by Gaussian elimination:

1. $x + 4y - z = -5, x + y - 6z = -12, 3x - y - z = 4$

   (Answer: Unique Solution, $x = \frac{117}{71}, y = -\frac{81}{71}, z = \frac{148}{71}$)

2. $x_1 - x_2 + x_3 + x_4 = 6, 2x_1 - x_3 - x_4 = 5, 2x_1 - 2x_2 + x_4 = 4, x_2 + x_3 - x_4 = 3$

   (Answer: Unique Solution, $x_1 = \frac{17}{3}, x_2 = 6, x_3 = \frac{5}{3}, x_4 = \frac{14}{3}$)

3. $x + 2y - 3z + 2t = 3, 2x + 3y - 6z - t = 1, x - y - z + t = 4$

   (Answer: Infinite Solutions, $x = 17 - 13t, y = 5 - 5t, z = 8 - 7t$)

4. $x_1 + 2x_2 + 3x_3 = 3, 2x_1 + 3x_2 + x_3 = 1, x_1 + x_2 - 2x_3 = -2$

   (Answer: Infinite Solutions, $x_1 = 7x_3 - 7, x_2 = 5 - 5x_3$)

5. $2x - 3y + z = -1, x + 4y + 5z = 25, 3x + y + 6z = 15$

   (Answer: No Solution)

Program 4

Program to solve system of algebraic equations using Gauss-Jordan Method.

Aim: To solve given system of linear algebraic equations using Gauss-Jordan method using Mathematics Softwares (FOSS).

Software: Maxima

Keys:

Key	Function
kill (all)	Unbinds all items on all infolists
linsolve_params	When linsolve_params is true, linsolve also generates the %r symbols used to represent arbitrary parameters. Otherwise, linsolve solves an under-determined system of equations with some variables expressed in terms of others.
linsolve ([expr_1, …, expr_m], [x_1, …, x_n])	Solves the list of simultaneous linear equations for the list of variables. The expressions must each be polynomials in the variables and may be equations.
length (expr)	Returns (by default) the number of parts in the external (displayed) form of *expr*. For lists this is the number of elements.
pop (list)	pop removes and returns the first element from the list list.
coefmatrix ([eqn_1, …, eqn_m], [x_1, …, x_n])	Returns the coefficient matrix for the variables x_1, …, x_n of the system of linear equations eqn_1, …, eqn_m.
augcoefmatrix ([eqn_1, …, eqn_m], [x_1, …, x_n])	Returns the augmented coefficient matrix for the variables x_1, …, x_n of the system of linear equations eqn_1, …, eqn_m.
triangularize (M)	Returns the upper triangular form of the matrix M, as produced by Gaussian elimination
list_matrix_entries (M)	Returns a list containing the elements of the matrix M
matrix_size *(M)*	Return a two-member list that gives the number of rows and columns, respectively of the matrix M.
and	The logical conjunction operator.
if cond_1 *then* expr_1 *else* expr_0	evaluates to expr_1 *if* cond_1 evaluates to true, otherwise the expression evaluates to expr_0.
print ("*text*", *expr*)$	Displays *text* within inverted commas and evaluates and displays *expr*
echelon (M)	Returns the echelon form of the matrix M, as produced by Gaussian elimination.
col *(M, i)*	Returns the *i*'th column of the matrix *M*.

Note:1. Press Shift+Enter for evaluation of commands and display of output.
2. Replace semicolon (;) by dollar ($) to suppress output of any input line and vice-versa.
3. Start each session with kill(all)$ or quit()$ to remove previously assigned values of all symbols

Definitions and Formulae:

Gauss-Jordan Method: It is one of the direct methods of solving a system of linear algebraic equations $\boldsymbol{AX} = \boldsymbol{B}$. It is improved form of Gaussian elimination. In this method, the augmented matrix $[\boldsymbol{A}|\boldsymbol{B}]$ is reduced by a series of row operations to reduced row echelon form $[\boldsymbol{I}|\boldsymbol{C}]$ which readily gives the solution $\boldsymbol{X} = \boldsymbol{C}$. If the coefficient matrix $\boldsymbol{A}$ is square invertible, then Gauss-Jordan method reduces $\boldsymbol{A}$ to the identity matrix $\boldsymbol{I}$. This method is also used to find the inverse of an invertible matrix.

Program:

Program to find augmented matrix, reduced echelon form of augmented matrix and hence test consistency of given system of linear algebraic equations $AX = B$ and find solution if consistent using Gauss-Jordan Method.

```
kill(all)$
rref(A):=block([p,q,k],[p,q]:matrix_size(A),A:echelon(A),k:min(p,q),
   for I thru min(p,q) do (if A[i,i]=0 then (k:i-1,return())),
   for i:k thru 2 step -1 do (for j from i-1 thru 1 step -1 do A:rowop(A,j,i,A[j,i])),A)$
linsolve_params:false$
EQ:[given system of equations separated by comma]$
X:[variables separated comma]$
S:linsolve(EQ,X)$
A:augcoefmatrix (EQ, X)$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Augmented Matrix is", A)$
print("Normal form of Augmented Matrix is", rref(A))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print("Required solution is") and
while S # [ ] do disp(pop(S))
else print("Given system has infinite solutions") and
print("Required solution is") and
while S # [ ] do disp(pop(S))$
```

Worked Examples:

Problem 1. Write a program to test consistency and find solution if consistent of given system of equations by Gauss-Jordan Method:

$$4x_1 - x_2 = 1, -x_1 + 4x_2 - x_3 = 0, -x_2 + 4x_3 - x_4 = 0, -x_3 + 4x_4 = 0$$

Program:

```
kill(all)$
rref(A):=block([p,q,k],[p,q]:matrix_size(A),A:echelon(A),k:min(p,q),
   for i thru min(p,q) do (if A[i,i]=0 then (k:i−1,return())),
   for i:k thru 2 step −1 do (for j from i−1 thru 1 step −1 do A:rowop(A,j,i,A[j,i])),A)$
linsolve_params:false$
EQ:[4·x[1]−x[2]=1,−x[1]+4·x[2]−x[3]=0,−x[2]+4·x[3]−x[4]=0,−x[3]+4·x[4]=0]$
X:[x[1],x[2],x[3],x[4]]$
S:linsolve(EQ,X)$
A:augcoefmatrix (EQ, X)$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Augmented Matrix is", A)$
print("Normal form of Augmented Matrix is", rref(A))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print("Required solution is") and
while S # [ ] do disp(pop(S))
else print("Given system has infinite solutions") and
print("Required solution is") and
while S # [ ] do disp(pop(S))$
```

Output:

Given System of Equations is

$4\,x_1 - x_2 = 1$

$-x_3 + 4\,x_2 - x_1 = 0$

$-x_4 + 4\,x_3 - x_2 = 0$

$4\,x_4 - x_3 = 0$

Augmented Matrix is $\begin{pmatrix} 4 & -1 & 0 & 0 & -1 \\ -1 & 4 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 4 & 0 \end{pmatrix}$

Normal form of Augmented Matrix is $\begin{pmatrix} 1 & 0 & 0 & 0 & -\frac{56}{209} \\ 0 & 1 & 0 & 0 & -\frac{15}{209} \\ 0 & 0 & 1 & 0 & -\frac{4}{209} \\ 0 & 0 & 0 & 1 & -\frac{1}{209} \end{pmatrix}$

Given system has unique solution

Required solution is

$x_1 = \frac{56}{209}$

$x_2 = \frac{15}{209}$

$x_3 = \frac{4}{209}$

$x_4 = \frac{1}{209}$

Problem 2. Write a program to test consistency and find solution if consistent of given system of equations by Gauss-Jordan Method:

$$x + 2y - 3z + 2t = 3, 2x + 3y - 6z - t = 1, x - y - z + t = 4, x - z + 6t = 9$$

Program:

```
kill(all)$
rref(A):=block([p,q,k],[p,q]:matrix_size(A),A:echelon(A),k:min(p,q),
   for i thru min(p,q) do (if A[i,i]=0 then (k:i−1,return())),
   for i:k thru 2 step −1 do (for j from i−1 thru 1 step −1 do A:rowop(A,j,i,A[j,i])),A)$
linsolve_params:false$
EQ:[x+2·y−3·z+2·t=3,2·x+3·y−6·z−t=1,x−y−z+t=4,x−z+6·t=9]$
X:[x,y,z,t]$
S:linsolve(EQ,X)$
A:augcoefmatrix (EQ, X)$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Augmented Matrix is", A)$
print("Normal form of Augmented Matrix is", rref(A))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print("Required solution is") and
while S # [ ] do disp(pop(S))
else print("Given system has infinite solutions") and
print("Required solution is") and
while S # [ ] do disp(pop(S))$
```

Output:

solve: dependent equations eliminated: (1)

Given System of Equations is

$-3z+2y+x+2t=3$

$-6z+3y+2x-t=1$

$-z-y+x+t=4$

$-z+x+6t=9$

Augmented Matrix is $\begin{pmatrix} 1 & 2 & -3 & 2 & -3 \\ 2 & 3 & -6 & -1 & -1 \\ 1 & -1 & -1 & 1 & -4 \\ 1 & 0 & -1 & 6 & -9 \end{pmatrix}$

Normal form of Augmented Matrix is $\begin{pmatrix} 1 & 0 & 0 & 13 & -17 \\ 0 & 1 & 0 & 5 & -5 \\ 0 & 0 & 1 & 7 & -8 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

Given system has infinite solutions

Required solution is

$x=17-13t$

$y=5-5t$

$z=8-7t$

Problem 3. Write a program to test consistency and find solution if consistent of given system of equations by Gauss-Jordan Method:

$$2x - 3y + z = -1, x + 4y + 5z = 2, 3x + y + 6z = 15$$

Program:

```
kill(all)$
rref(A):=block([p,q,k],[p,q]:matrix_size(A),A:echelon(A),k:min(p,q),
   for i thru min(p,q) do (if A[i,i]=0 then (k:i−1,return())),
   for i:k thru 2 step −1 do (for j from i−1 thru 1 step −1 do A:rowop(A,j,i,A[j,i])),A)$
linsolve_params:false$
EQ:[2·x−3·y+z=−1,x+4·y+5·z=2,3·x+y+6·z=15]$
X:[x,y,z]$
S:linsolve(EQ,X)$
A:augcoefmatrix (EQ, X)$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Augmented Matrix is", A)$
print("Normal form of Augmented Matrix is", rref(A))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print("Required solution is") and
while S # [ ] do disp(pop(S))
else print("Given system has infinite solutions") and
print("Required solution is") and
while S # [ ] do disp(pop(S))$
```

Output: *Given System of Equations is*

$z-3\,y+2\,x=-1$

$5\,z+4\,y+x=2$

$6\,z+y+3\,x=15$

Augmented Matrix is $\begin{pmatrix} 2 & -3 & 1 & 1 \\ 1 & 4 & 5 & -2 \\ 3 & 1 & 6 & -15 \end{pmatrix}$

Normal form of Augmented Matrix is $\begin{pmatrix} 1 & 0 & \frac{19}{11} & -\frac{2}{11} \\ 0 & 1 & \frac{9}{11} & -\frac{5}{11} \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Given system has no solution

Problem 4. Write a program to test consistency and find solution if consistent of given system of equations by Gauss-Jordan Method:

$$x + 2y - 3z = 7,\ x + 8y - 5z = 16,\ 2x - y + 2z = 5,\ 3x - 5y + z = 3$$

Program:

```
kill(all)$
rref(A):=block([p,q,k],[p,q]:matrix_size(A),A:echelon(A),k:min(p,q),
   for i thru min(p,q) do (if A[i,i]=0 then (k:i−1,return())),
   for i:k thru 2 step −1 do (for j from i−1 thru 1 step −1 do A:rowop(A,j,i,A[j,i])),A)$
linsolve_params:false$
EQ:[x+2·y−3·z=7,x+8·y−5·z=16,2·x−y+2·z=5,3·x−5·y+z=3]$
X:[x,y,z]$
S:linsolve(EQ,X)$
A:augcoefmatrix (EQ, X)$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Augmented Matrix is", A)$
print("Normal form of Augmented Matrix is", rref(A))$
if length(S)=0 then
print("Given system has no solution")
elseif length(S)=length(X) then
print("Given system has unique solution") and
print("Required solution is") and
while S # [ ] do disp(pop(S))
else print("Given system has infinite solutions") and
print("Required solution is") and
while S # [ ] do disp(pop(S))$
```

solve: dependent equations eliminated: (4)

Output:

Given System of Equations is

$-3z+2y+x=7$

$-5z+8y+x=16$

$2z-y+2x=5$

$z-5y+3x=3$

Augmented Matrix is $\begin{pmatrix} 1 & 2 & -3 & -7 \\ 1 & 8 & -5 & -16 \\ 2 & -1 & 2 & -5 \\ 3 & -5 & 1 & -3 \end{pmatrix}$

Normal form of Augmented Matrix is $\begin{pmatrix} 1 & 0 & 0 & -\frac{131}{38} \\ 0 & 1 & 0 & -\frac{27}{19} \\ 0 & 0 & 1 & \frac{9}{38} \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Given system has unique solution

Required solution is

$x=\frac{131}{38}$

$y=\frac{27}{19}$

$z=-\frac{9}{38}$

Exercise:

Write a program to test consistency and find solution if consistent of given system of equations by Gauss-Jordan Method:

1. $x + y - z = 2, 2x + 3y + 5z = -3, x + 2y - 5z = 6$

   (Answer: Unique Solution $x = 1, y = 0, z = -1$)

2. $2x + y + z = 10, 3x + 2y + 3z = 18, x + 4y + 9z = 16$

   (Answer: Unique Solution $x = 7, y = -9, z = 5$)

3. $x_1 - x_2 + x_3 + x_4 = 6, 2x_1 - x_3 - x_4 = 5, 2x_1 - 2x_2 + x_4 = 4, x_2 + x_3 - x_4 = 3$

   (Answer: Unique Solution $x_1 = \frac{17}{3}, x_2 = 6, x_3 = \frac{5}{3}, x_4 = \frac{14}{3}$)

4. $5x_1 + x_2 + x_3 + x_4 = 4, \quad x_1 + 7x_2 + x_3 + x_4 = 12,$
   $x_1 + x_2 + 6x_3 + x_4 = -5, \quad x_1 + x_2 + x_3 + 4x_4 = -6$

   (Answer: Unique Solution $x_1 = 1, x_2 = 2, \ x_3 = -1, \ x_4 = -2$)

5. $x_1 + 2x_2 + 3x_3 = 3, 2x_1 + 3x_2 + x_3 = 1, x_1 + x_2 - 2x_3 = -2$

   (Answer: Infinite Solutions $x_1 = 7x_3 - 7, x_2 = 5 - 5x_3$)

6. $x + y = 2, 2x + 5z = -3, x + 2y - 5z = 6$

   (Answer: Unique Solution $\mathrm{x} = -1, \mathrm{y} = 3, \mathrm{z} = -\frac{1}{5}$)

7. $2x - 3y + z = -1, x + 4y + 5z = 25, 3x + y + 6z = 15$

   (Answer: No Solution)

Program 5

Program to solve system of algebraic equations using Gauss-Jacobi Method.

Aim: To find a solution of given system of linear algebraic equations from initial guess by Gauss-Jacobi Method using Mathematics Softwares (FOSS).

Software: Maxima

Keys:

Key	**Function**
kill (all)	Unbinds all items on all infolists
float *(expr)*	Converts integers, rational numbers and bigfloats in *expr* to floating point numbers
numer:true	numer causes some mathematical functions (including exponentiation) with numerical arguments to be evaluated in floating point. Default value is false.
fpprintprec	This is an option variable to decide the number of digits to print when printing an ordinary float or bigfloat number. Default value is 16. Set any integer from 2 to 16.
:=	The function definition operator
define *(f(x_1, …, x_n), expr)*	Defines a function named *f* with arguments *x_1*, …, *x_n* and function body *expr*.
memoizing function *f*[*x_1*, ..., *x_n*] := *expr*	A memoizing function caches the result the first time it is called with a given argument, and returns the stored value, without recomputing it, when that same argument is given.
[a_1, a_2,…,a_m]	List of numbers/objects a_1, a_2,…,a_m.
if cond_1 *then* expr_1 *else* expr_0	evaluates to expr_1 *if* cond_1 evaluates to true, otherwise the expression evaluates to expr_0.
print (“*text*”, *expr*)$	Displays *text* within inverted commas and evaluates and displays *expr*
block ([v_1, …, v_m], expr_1, …, expr_n)	The function *block* allows to make the variables *v_1*, …, *v_m* to be local for a sequence of commands.
push *(item, list)*	*push* prepends the item *item* to the list *list* and returns a copy of the new list
pop (list)	pop removes and returns the first element from the list list.
reverse *(list)*	Reverses the order of the members of the *list* (not the members themselves)
table_form()	Displays a 2D list in a form that is more readable than the output from *Maxima*’s default output routine. The input is a list of one or more lists.
<=	less than or equal to
L[i]	Subscript operator for L_i
diff (expr, x)	Returns the first derivative of expr with respect to the variable x

Note:1. Press Shift+Enter for evaluation of commands and display of output.

2. Replace semicolon (;) by dollar ($) to suppress output of any input line and vice-versa.
3. Start each session with kill(all)$ or quit()$ to remove previously assigned values of all symbols

Definitions and Formulae:

Diagonally dominant system: A square matrix $A = [a_{ij}]_{n\times n}$ is said to be diagonally dominant if for every row of the matrix, the magnitude of the diagonal entry in a row is larger or equal to the sum of all the other (non-diagonal) entries in that row. In other words,

$$|a_{ii}| \geq \sum_{j\neq i} |a_{ij}| \ \forall\, i$$

A square matrix is said to be strictly diagonally dominant if

$$|a_{ii}| > \sum_{j\neq i} |a_{ij}| \ \forall\, i$$

A square Linear system $\boldsymbol{AX} = \boldsymbol{B}$ is said to be (strictly) diagonally dominant if $\boldsymbol{A}$ is (strictly) diagonally dominant matrix.

Gauss-Jacobi Iteration Method: It is one of the iterative methods for solving square linear system $\boldsymbol{AX} = \boldsymbol{B}$ where $\boldsymbol{A}$ is a square matrix. Diagonal entries, also called pivot elements of $\boldsymbol{A}$, are assumed to be non-zero. In this method, the coefficient matrix $\boldsymbol{A}$ is decomposed as $\boldsymbol{A} = \boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U}$ where $\boldsymbol{L}$ is strictly lower triangular, $\boldsymbol{D}$ is diagonal, $\boldsymbol{U}$ is strictly upper triangular parts of $\boldsymbol{A}$. Then, given system is solved for diagonal entries to get:

$$\boldsymbol{AX} = \boldsymbol{B}$$

$$(\boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U})\boldsymbol{X} = \boldsymbol{B}$$

$$\boldsymbol{DX} = \boldsymbol{B} - (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{X}$$

$$\boldsymbol{X} = \boldsymbol{D}^{-1}[\boldsymbol{B} - (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{X}]$$

Assuming initial approximation as $\boldsymbol{X} = \boldsymbol{X}^{(0)}$ and substituting it in the RHS of above equation we get the first approximation as

$$\boldsymbol{X}^{(1)} = \boldsymbol{D}^{-1}\left[\boldsymbol{B} - (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{X}^{(0)}\right]$$

Then successive approximations are given by

$$\boldsymbol{X}^{(k+1)} = \boldsymbol{D}^{-1}\left[\boldsymbol{B} - (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{X}^{(k)}\right]$$

If given system is

$$\begin{array}{l} \boldsymbol{a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1} \\ \boldsymbol{a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2} \\ \vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\ \boldsymbol{a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n} \end{array}$$

Then

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x_1} \\ \boldsymbol{x_2} \\ \vdots \\ \boldsymbol{x_n} \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \vdots \\ \boldsymbol{b_n} \end{bmatrix}$$

So that

$$L = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ a_{21} & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 \\ 0 & a_{22} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{nn} \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & 0 & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

Hence $\boldsymbol{X} = \boldsymbol{D^{-1}}[\boldsymbol{B} - (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{X}]$ becomes

$$\boldsymbol{x_i} = \frac{\boldsymbol{1}}{\boldsymbol{a_{ii}}}\left[\boldsymbol{b_i} - (\boldsymbol{a_{i1}x_1} + \boldsymbol{a_{i2}x_2} + \boldsymbol{a_{i3}x_3} + \cdots + \boldsymbol{a_{i(i-1)}x_{i-1}} + \boldsymbol{a_{i(i+1)}x_{i+1}} + \cdots + \boldsymbol{a_{in}x_n})\right]$$

$$\boldsymbol{i.e.}, \boldsymbol{x_i} = \frac{\boldsymbol{1}}{\boldsymbol{a_{ii}}}\left[\boldsymbol{b_i} - \sum_{\substack{j=1 \\ j\neq i}}^{n} \boldsymbol{a_{ij}x_j}\right], \boldsymbol{i} = \boldsymbol{1, 2, 3, \dots n}$$

If $\boldsymbol{X^{(0)}} = (\boldsymbol{x_1}^{(0)}, \boldsymbol{x_2}^{(0)}, \boldsymbol{x_3}^{(0)}, \dots, \boldsymbol{x_n}^{(0)})$ is the initial approximation then first and successive approximations are given by

$$\boldsymbol{x_i}^{(1)} = \frac{\boldsymbol{1}}{\boldsymbol{a_{ii}}}\left[\boldsymbol{b_i} - \sum_{\substack{j=1 \\ j\neq i}}^{n} \boldsymbol{a_{ij}x_j}^{(0)}\right]$$

and

$$\boldsymbol{x_i}^{(k+1)} = \frac{\boldsymbol{1}}{\boldsymbol{a_{ii}}}\left[\boldsymbol{b_i} - \sum_{\substack{j=1 \\ j\neq i}}^{n} \boldsymbol{a_{ij}x_j}^{(k)}\right], \boldsymbol{i} = \boldsymbol{1, 2, 3, \dots n}$$

This method is called the method of simultaneous displacement as old values of all variables in the RHS are simultaneously replaced by new values in the beginning of each iteration. If $\boldsymbol{AX} = \boldsymbol{B}$ is **strictly diagonally dominant system** then for **any choice of initial guess** $\boldsymbol{X^{(0)}}$ the Jacobi method **converges** to the unique solution. So, one can take $\boldsymbol{X^{(0)}} = (\boldsymbol{0}, \boldsymbol{0}, \dots, \boldsymbol{0})$ if no initial approximation is given. The iteration process may be continued for specified number of iterations or till roots of desired accuracy are obtained.

Program:

Program to find the solution of given linear system of equations

$$a_{11}x + a_{12}y + a_{13}z = b_1$$

$$a_{21}x + a_{22}y + a_{23}z = b_2$$

$$a_{31}x + a_{32}y + a_{33}x = b_3$$

with initial guess

$$x^{(0)} = y^{(0)} = z^{(0)} = 0$$

by Gauss-Jacobi Method by performing n **iterations.**

Solving given system for diagonal entries, we get Jacobi iteration scheme as

$$x^{(i)} = \frac{1}{a_{11}}\left[b_1 - a_{12}y^{(i-1)} - a_{13}z^{(i-1)}\right]$$

$$y^{(i)} = \frac{1}{a_{22}}\left[b_2 - a_{21}x^{(i-1)} - a_{23}z^{(i-1)}\right]$$

$$z^{(i)} = \frac{1}{a_{33}}\left[b_3 - a_{31}x^{(i-1)} - a_{32}y^{(i-1)}\right]$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[a11*x+a12*y+a13*z=b1,a21*x+a22*y+a23*z=b2,a31*x+a32*y+a33*z=b3]$
print(“Given System of Equations is”)$
while EQ # [ ] do disp(pop(EQ))$
print(“Gauss-Jacobi Iteration scheme for given system is”)$
x[i]:=(b1-a12*y[i-1]-a13*z[i-1])/a11;
y[i]:=(b2-a21*x[i-1]-a23*z[i-1])/a22;
z[i]:=(b3-a31*x[i-1]-a32*y[i-1])/a33;
x[0]:0$
y[0]:0$
z[0]:0$
n:given number of iterations$
print(“Initial Approximation is “,’x[0]=x[0],’y[0]=y[0],’z[0]=z[0])$
N:[[“Iteration No.”,”x”,”y”,”z”]]$
for i:1 thru n do N:push([i,x[i],y[i],z[i]],N)$
table_form(reverse(N))$
print(“Solution by Gauss-Jacobi Method is x=”,x[n], “y=”,y[n],”z=”,z[n])$
```

Program:

Program to find the solution of given linear system of equations

$$a_{11}x + a_{12}y + a_{13}z = b_1$$
$$a_{21}x + a_{22}y + a_{23}z = b_2$$
$$a_{31}x + a_{32}y + a_{33}x = b_3$$

with initial guess

$$x^{(0)} = y^{(0)} = z^{(0)} = 0$$

by Gauss-Jacobi Method for **given accuracy**.

Solving given system for diagonal entries, we get Jacobi iteration scheme as

$$x^{(i)} = \frac{1}{a_{11}}\left[b_1 - a_{12}y^{(i-1)} - a_{13}z^{(i-1)}\right]$$
$$y^{(i)} = \frac{1}{a_{22}}\left[b_2 - a_{21}x^{(i-1)} - a_{23}z^{(i-1)}\right]$$
$$z^{(i)} = \frac{1}{a_{33}}\left[b_3 - a_{31}x^{(i-1)} - a_{32}y^{(i-1)}\right]$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[a11*x+a12*y+a13*z=b1,a21*x+a22*y+a23*z=b2,a31*x+a32*y+a33*z=b3]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Gauss-Jacobi Iteration scheme for given system is")$
x[i]:=(b1-a12*y[i-1]-a13*z[i-1])/a11;
y[i]:=(b2-a21*x[i-1]-a23*z[i-1])/a22;
z[i]:=(b3-a31*x[i-1]-a32*y[i-1])/a33;
x[0]:0$
y[0]:0$
z[0]:0$
i:1$
accuracy:0.00001$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
N:[["Iteration No.","x","y","z"]]$
block(loop,N:push([i,x[i],y[i],z[i]],N),
   if abs(x[i]-x[i-1])<=accuracy and abs(y[i]-y[i-1])<=accuracy and
   abs(z[i]-z[i-1])<=accuracy then (table_form(reverse(N)),
      print("Solution by Gauss-Jacobi Method is x=",x[i], "y=",y[i],"z=",z[i]))
else(i:i+1,go(loop)))$
```

Note: You may take **accuracy:0.001, 0.0001, 0.00001** to get approximate root correct to **2 decimal places, 3 decimal places, 4 decimal places** respectively.

Worked Examples:

Problem 1. Write a program to find the solution of given linear system of equations with initial guess by Gauss-Jacobi Method performing 13 iterations.

$$10x + y + z = 12;\ 2x + 10y + z = 13;\ 2x + 2y + 10z = 14$$

$$x^{(0)} = y^{(0)} = z^{(0)} = 0$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[10·x+y+z=12,2·x+10·y+z=13,2·x+2·y+10·z=14]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Gauss-Jacobi Iteration scheme for given system is")$
x[i]:=(12−y[i−1]−z[i−1])/10;
y[i]:=(13−2·x[i−1]−z[i−1])/10;
z[i]:=(14−2·x[i−1]−2·y[i−1])/10;
x[0]:0$
y[0]:0$
z[0]:0$
n:13$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
N:[["Iteration No.","x","y","z"]]$
for i:1 thru n do N:push([i,x[i],y[i],z[i]],N)$
table_form(reverse(N))$
print("Solution by Gauss-Jacobi Method is x=",x[n], "y=",y[n],"z=",z[n])$
```

Output:

Given System of Equations is

$z+y+10\,x=12$

$z+10\,y+2\,x=13$

$10\,z+2\,y+2\,x=14$

Gauss−Jacobi Iteration scheme for given system is

$$x_i := \frac{12-y_{i-1}-z_{i-1}}{10}$$

$$y_i := \frac{13-2\,x_{i-1}-z_{i-1}}{10}$$

$$z_i := \frac{14-2\,x_{i-1}+(-2)\,y_{i-1}}{10}$$

Initial Approximation is $x_0=0\ \ y_0=0\ \ z_0=0$

Iteration No.	x	y	z
1	1.2	1.3	1.4
2	0.93	0.92	0.9
3	1.018	1.024	1.03
4	0.9946	0.9934	0.9916
5	1.0015	1.00192	1.0024
6	0.999568	0.99946	0.999316
7	1.000122	1.000155	1.000194
8	0.9999651	0.9999561	0.9999446
9	1.00001	1.000013	1.000016
10	0.9999972	0.9999964	0.9999955
11	1.000001	1.000001	1.000001
12	0.9999998	0.9999997	0.9999996
13	1.0	1.0	1.0

Solution by Gauss−Jacobi Method is $x=\ 1.0\ \ y=\ 1.0\ \ z=\ 1.0$

Problem 2. Write a program to find the solution of given linear system of equations with initial guess by Gauss-Jacobi Method performing 16 iterations.

$$5x - y + 3z = 10;\ 3x + 6y = 18;\ x + y + 5z = -10$$

$$x^{(0)} = 3, \quad y^{(0)} = 0, \quad z^{(0)} = -2$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[5·x−y+3·z=10,3·x+6·y=18,x+y+5·z=−10]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Gauss-Jacobi Iteration scheme for given system is")$
x[i]:=(10+y[i−1]−3·z[i−1])/5;
y[i]:=(18−3·x[i−1])/6;
z[i]:=(−10−x[i−1]−y[i−1])/5;
x[0]:3$
y[0]:0$
z[0]:−2$
n:16$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
N:[["Iteration No.","x","y","z"]]$
for i:1 thru n do N:push([i,x[i],y[i],z[i]],N)$
table_form(reverse(N))$
print("Solution by Gauss-Jacobi Method is x=",x[n], "y=",y[n],"z=",z[n])$
```

Output:

Given System of Equations is

$3z - y + 5x = 10$

$6y + 3x = 18$

$5z + y + x = -10$

Gauss−Jacobi Iteration scheme for given system is

$$x_i := \frac{10 + y_{i-1} + (-3)\,z_{i-1}}{5}$$

$$y_i := \frac{18 - 3x_{i-1}}{6}$$

$$z_i := \frac{-10 - x_{i-1} - y_{i-1}}{5}$$

Initial Approximation is $x_0 = 3 \quad y_0 = 0 \quad z_0 = -2$

Iteration No.	*x*	*y*	*z*
1	3.2	1.5	−2.6
2	3.86	1.4	−2.94
3	4.044	1.07	−3.052
4	4.0452	0.978	−3.0228
5	4.00928	0.9774	−3.00464
6	3.998264	0.99536	−2.997336
7	3.997474	1.000868	−2.998725
8	3.999408	1.001263	−2.999668
9	4.000054	1.000296	−3.000134
10	4.00014	0.9999732	−3.00007
11	4.000037	0.9999301	−3.000023
12	4.0	0.9999817	−2.999993
13	3.999992	1.0	−2.999996
14	3.999998	1.000004	−2.999999
15	4.0	1.000001	−3.0
16	4.0	1.0	−3.0

Solution by Gauss−Jacobi Method is x= 4.0 *y=* 1.0 *z=* −3.0

Problem 3. Write a program to find the solution of given linear system of equations with initial guess by Gauss-Jacobi Method correct to 5 decimal places.

$$20x + y - 2z = 17;\ 3x + 20y - z = -18;\ 2x - 3y + 20z = 25$$

$$x^{(0)} = 0, \quad y^{(0)} = 0, \quad z^{(0)} = 0$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[20·x+y−2·z=17,3·x+20·y−z=−18,2·x−3·y+20·z=25]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Gauss-Jacobi Iteration scheme for given system is")$
x[i]:=(17−y[i−1]+2·z[i−1])/20;
y[i]:=(−18−3·x[i−1]+z[i−1])/20;
z[i]:=(25−2·x[i−1]+3·y[i−1])/20;
x[0]:0$
y[0]:0$
z[0]:0$
i:1$
accuracy:0.000001$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
N:[["Iteration No.","x","y","z"]]$
block(loop,N:push([i,x[i],y[i],z[i]],N),
  if abs(x[i]−x[i−1])<=accuracy and abs(y[i]−y[i−1])<=accuracy and
  abs(z[i]−z[i−1])<=accuracy then (table_form(reverse(N)),
    print("Solution by Gauss-Jacobi Method is x=",x[i], "y=",y[i],"z=",z[i]))
else(i:i+1,go(loop)))$
```

Output:

Given System of Equations is

$-2z+y+20x=17$

$-z+20y+3x=-18$

$20z-3y+2x=25$

Gauss−Jacobi Iteration scheme for given system is

$$x_i := \frac{17 - y_{i-1} + 2z_{i-1}}{20}$$

$$y_i := \frac{-18 - 3x_{i-1} + z_{i-1}}{20}$$

$$z_i := \frac{25 - 2x_{i-1} + 3y_{i-1}}{20}$$

Initial Approximation is $x_0 = 0 \quad y_0 = 0 \quad z_0 = 0$

Iteration No.	*x*	*y*	*z*
1	0.85	−0.9	1.25
2	1.02	−0.965	1.03
3	1.00125	−1.0015	1.00325
4	1.0004	−1.000025	0.99965
5	0.9999662	−1.000078	0.9999563
6	0.9999995	−0.9999971	0.9999918
7	0.999999	−1.0	1.0
8	1.0	−0.9999998	1.0
9	1.0	−1.0	1.0

Solution by Gauss−Jacobi Method is $x=$ 1.0 $y=$ −1.0 $z=$ 1.0

Problem 4. Write a program to find the solution of given linear system of equations with initial guess by Gauss-Jacobi Method performing 10 iterations. What is your observation about convergence of iteration?

$$2x - 6y + 8z = 24;\ 5x + 4y - 3z = 2;\ 3x + y + 2z = 16$$

$$x^{(0)} = 0, \quad y^{(0)} = 0, \quad z^{(0)} = 0$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[2·x−6·y+8·z=24,5·x+4·y−3·z=2,3·x+y+2·z=16]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Gauss-Jacobi Iteration scheme for given system is")$
x[i]:=(24+6·y[i−1]−8·z[i−1])/2;
y[i]:=(2−5·x[i−1]+3·z[i−1])/4;
z[i]:=(16−3·x[i−1]−y[i−1])/2;
x[0]:0$
y[0]:0$
z[0]:0$
n:10$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
N:[["Iteration No.","x","y","z"]]$
for i:1 thru n do N:push([i,x[i],y[i],z[i]],N)$
table_form(reverse(N))$
print("Solution by Gauss-Jacobi Method is x=",x[n], "y=",y[n],"z=",z[n])$
```

Output:

Given System of Equations is

$8\,z-6\,y+2\,x=24$

$-3\,z+4\,y+5\,x=2$

$2\,z+y+3\,x=16$

Gauss−Jacobi Iteration scheme for given system is

$$x_i:=\frac{24+6\,y_{i-1}+(-8)\,z_{i-1}}{2}$$

$$y_i:=\frac{2-5\,x_{i-1}+3\,z_{i-1}}{4}$$

$$z_i:=\frac{16-3\,x_{i-1}-y_{i-1}}{2}$$

Initial Approximation is $x_0=0$ $y_0=0$ $z_0=0$

Iteration No.	*x*	*y*	*z*
1	12	0.5	8
2	−18.5	−8.5	−10.25
3	27.5	15.9375	40.0
4	−100.1875	−3.875	−41.21875
5	165.25	94.82031	160.2188
6	−344.4141	−85.89844	−287.2852
7	903.4453	215.5537	567.5703
8	−1611.62	−703.1289	−1454.945
9	3722.393	923.8165	2776.995
10	−8324.529	−2569.745	−6037.497

Solution by Gauss−Jacobi Method is x= −8324.529 *y=* −2569.745 *z=* −6037.497

Observation: Clearly, Jacobi iteration is **not** convergent as the given system is **not** diagonally dominant

Exercise:

Write a program to find the solution of given linear system of equations with initial guess by Gauss-Jacobi Method.

1. $\mathbf{5x - y + z = 10;\ x + 2y = 6;\ x + y + 5z = -1}$

   $\mathbf{x^{(0)} = 2,\ y^{(0)} = 3,\ z^{(0)} = 0}$ by performing 10 iterations.

   (Answer: $x = 2.555549, y = 1.722232, z = -1.055549$)

2. $\mathbf{27x + 6y - z = 85, 6x + 15y + 2z = 72, x + y + 54z = 110}$

   $\mathbf{x^{(0)} = 0,\ y^{(0)} = 0,\ z^{(0)} = 0}$ correct to 4 decimal places.

   (Answer: $x = 2.425475, y = 3.573014, z = 1.925954$)

3. $\mathbf{5x + 2y + z = 12;\ x + 4y + 2z = 15;\ x + 2y + 5z = 20}$

   $\mathbf{x^{(0)} = 0,\ y^{(0)} = 0,\ z^{(0)} = 0}$ correct to 3 decimal places.

   (Answer: $x = 1.000024, y = 2.000027, z = 3.000024$)

4. $\mathbf{10x - 2y - z - t = 2}$

   $\mathbf{-2x + 10y - z - t = 14}$

   $\mathbf{-x - y + 10z - 2t = 25}$

   $\mathbf{-x - y - 2z + 10t = 1}$

   $\mathbf{x^{(0)} = 0,\ y^{(0)} = 0,\ z^{(0)} = 0, t^{(0)} = 0}$ correct to 6 decimal places.

   (Answer: $x = 1, y = 2, z = 3, t = 1$)

Program 6

Program to solve system of algebraic equations using Gauss-Seidel Method.

Aim: To find the solution of given system of linear algebraic equations from initial guess by Gauss-Seidel Method using Mathematics Softwares (FOSS).

Software: Maxima

Keys:

Key	**Function**
kill (all)	Unbinds all items on all infolists
float *(expr)*	Converts integers, rational numbers and bigfloats in *expr* to floating point numbers
numer:true	numer causes some mathematical functions (including exponentiation) with numerical arguments to be evaluated in floating point. Default value is false.
fpprintprec	This is an option variable to decide the number of digits to print when printing an ordinary float or bigfloat number. Default value is 16. Set any integer from 2 to 16.
:=	The function definition operator
define *(f(x_1, …, x_n), expr)*	Defines a function named *f* with arguments *x_1*, …, *x_n* and function body *expr*.
memoizing function *f*[*x_1*, ..., *x_n*] := *expr*	A memoizing function caches the result the first time it is called with a given argument, and returns the stored value, without recomputing it, when that same argument is given.
$[a_1, a_2,\ldots,a_m]$	List of numbers/objects $a_1, a_2,\ldots,a_m$.
if cond_1 *then* expr_1 *else* expr_0	evaluates to expr_1 *if* cond_1 evaluates to true, otherwise the expression evaluates to expr_0.
print (“*text*”, *expr*)$	Displays *text* within inverted commas and evaluates and displays *expr*
block ([v_1, …, v_m], expr_1, …, expr_n)	The function *block* allows to make the variables *v_1*, …, *v_m* to be local for a sequence of commands.
push *(item, list)*	*push* prepends the item *item* to the list *list* and returns a copy of the new list
pop (list)	pop removes and returns the first element from the list list.
reverse *(list)*	Reverses the order of the members of the *list* (not the members themselves)
table_form()	Displays a 2D list in a form that is more readable than the output from *Maxima*’s default output routine. The input is a list of one or more lists.
<=	less than or equal to
L[i]	Subscript operator for L_i
diff (expr, x)	Returns the first derivative of expr with respect to the variable x

Note:1. Press Shift+Enter for evaluation of commands and display of output.
2. Replace semicolon (;) by dollar ($) to suppress output of any input line and vice-versa.
3. Start each session with kill(all)$ or quit()$ to remove previously assigned values of all symbols

Definitions and Formulae:

Gauss-Seidel Iteration Method: It is a modified Gauss-Jacobi iteration method for solving square linear system $\boldsymbol{AX}=\boldsymbol{B}$ where $\boldsymbol{A}$ is a square matrix. In this method, like Jacobi method, the coefficient matrix $\boldsymbol{A}$ is decomposed as $\boldsymbol{A}=\boldsymbol{L}+\boldsymbol{D}+\boldsymbol{U}$ where $\boldsymbol{L}$ is strictly lower triangular, $\boldsymbol{D}$ is diagonal, $\boldsymbol{U}$ is strictly upper triangular parts of $\boldsymbol{A}$. Then, given system is solved for diagonal entries to get:

$$\boldsymbol{X}=\boldsymbol{D}^{-1}[\boldsymbol{B}-(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{X}]$$

If $\boldsymbol{X}=(\boldsymbol{x_1},\boldsymbol{x_2},\boldsymbol{x_3},\dots,\boldsymbol{x_n})$ then above equation becomes

$$\boldsymbol{x_i}=\frac{\boldsymbol{1}}{\boldsymbol{a_{ii}}}\left[\boldsymbol{b_i}-\sum_{\substack{j=1\\ j\neq i}}^{n}\boldsymbol{a_{ij}x_j}\right],\boldsymbol{i}=\boldsymbol{1},\boldsymbol{2},\boldsymbol{3},\dots\boldsymbol{n}$$

Assuming initial approximation $\boldsymbol{X}^{(0)}=(\boldsymbol{x_1}^{(0)},\boldsymbol{x_2}^{(0)},\ \boldsymbol{x_3}^{(0)},\dots,\boldsymbol{x_n}^{(0)})$, $\boldsymbol{x_1}^{(1)}$ is computed using $(\boldsymbol{x_2}^{(0)},\ \boldsymbol{x_3}^{(0)},\dots,\boldsymbol{x_n}^{(0)})$ in the RHS of above equation. While computing $\boldsymbol{x_2}^{(1)}$, $(\boldsymbol{x_1}^{(1)},\ \boldsymbol{x_3}^{(0)},\dots,\boldsymbol{x_n}^{(0)}$ is used. **Note that new value $\boldsymbol{x_1}^{(1)}$ of $\boldsymbol{x_1}$ is immediately used in calculating $\boldsymbol{x_2}^{(1)}$. Similarly, $\boldsymbol{x_i}^{(1)}$ is calculated using latest available values $\boldsymbol{x_1}^{(1)},\ \boldsymbol{x_2}^{(1)},\dots,\ \boldsymbol{x_{i-1}}^{(1)}$.** Thus Gauss-Seidel iteration scheme for first approximation is given by

$$\boldsymbol{x_i}^{(1)}=\frac{\boldsymbol{1}}{\boldsymbol{a_{ii}}}\left[\boldsymbol{b_i}-\sum_{j=1}^{i-1}\boldsymbol{a_{ij}x_i}^{(1)}-\sum_{j=i+1}^{n}\boldsymbol{a_{ij}x_i}^{(0)}\right],\boldsymbol{i}=\boldsymbol{1},\boldsymbol{2},\boldsymbol{3},\dots\boldsymbol{n}$$

Then successive approximations are given by

$$\boldsymbol{x_i}^{(k+1)}=\frac{\boldsymbol{1}}{\boldsymbol{a_{ii}}}\left[\boldsymbol{b_i}-\sum_{j=1}^{i-1}\boldsymbol{a_{ij}x_i}^{(k+1)}-\sum_{j=i+1}^{n}\boldsymbol{a_{ij}x_i}^{(k)}\right],\boldsymbol{i}=\boldsymbol{1},\boldsymbol{2},\boldsymbol{3},\dots\boldsymbol{n}$$

In matrix form,

$$\boldsymbol{X}^{(k+1)}=\boldsymbol{D}^{-1}\left[\boldsymbol{B}-\boldsymbol{LX}^{(k+1)}-\boldsymbol{UX}^{(k)}\right]$$

This method is called the method of successive displacement as old values of variables in the RHS are successively replaced by new values as soon as they are available. If $\boldsymbol{AX}=\boldsymbol{B}$ is **strictly diagonally dominant system** then for **any choice of initial guess** $\boldsymbol{X}^{(0)}$ the Gauss-Seidel method **converges** to the unique solution. So, one can take $\boldsymbol{X}^{(0)}=(\boldsymbol{0},\boldsymbol{0},\dots,\boldsymbol{0})$ if no initial approximation is given. The iteration process may be continued for specified number of iterations or till roots of desired accuracy are obtained. Gauss-Seidel converges faster compared to Jacobi method.

Program:

Program to find the solution of given linear system of equations

$$a_{11}x + a_{12}y + a_{13}z = b_1$$
$$a_{21}x + a_{22}y + a_{23}z = b_2$$
$$a_{31}x + a_{32}y + a_{33}x = b_3$$

with initial guess

$$x^{(0)} = y^{(0)} = z^{(0)} = 0$$

by Gauss-Seidel Method by performing n **iterations.**

Solving given system for diagonal entries, we get Gauss-Seidel iteration scheme as

$$x^{(i)} = \frac{1}{a_{11}}\left[b_1 - a_{12}y^{(i-1)} - a_{13}z^{(i-1)}\right]$$
$$y^{(i)} = \frac{1}{a_{22}}\left[b_2 - a_{21}x^{(i)} - a_{23}z^{(i-1)}\right]$$
$$z^{(i)} = \frac{1}{a_{33}}\left[b_3 - a_{31}x^{(i)} - a_{32}y^{(i)}\right]$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[a11*x+a12*y+a13*z=b1,a21*x+a22*y+a23*z=b2,a31*x+a32*y+a33*z=b3]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Gauss-Seidel Iteration scheme for given system is")$
x[i]:=(b1-a12*y[i-1]-a13*z[i-1])/a11;
y[i]:=(b2-a21*x[i]-a23*z[i-1])/a22;
z[i]:=(b3-a31*x[i]-a32*y[i])/a33;
x[0]:0$
y[0]:0$
z[0]:0$
n:given number of iterations$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
N:[["Iteration No.","x","y","z"]]$
for i:1 thru n do N:push([i,x[i],y[i],z[i]],N)$
table_form(reverse(N))$
print("Solution by Gauss-Seidel Method is x=",x[n], "y=",y[n],"z=",z[n])$
```

Program:

Program to find the solution of given linear system of equations

$$a_{11}x + a_{12}y + a_{13}z = b_1$$
$$a_{21}x + a_{22}y + a_{23}z = b_2$$
$$a_{31}x + a_{32}y + a_{33}x = b_3$$

with initial guess

$$x^{(0)} = y^{(0)} = z^{(0)} = 0$$

by Gauss-Seidel Method for **given accuracy**.

Solving given system for diagonal entries, we get Gauss-Seidel iteration scheme as

$$x^{(i)} = \frac{1}{a_{11}}\left[b_1 - a_{12}y^{(i-1)} - a_{13}z^{(i-1)}\right]$$
$$y^{(i)} = \frac{1}{a_{22}}\left[b_2 - a_{21}x^{(i)} - a_{23}z^{(i-1)}\right]$$
$$z^{(i)} = \frac{1}{a_{33}}\left[b_3 - a_{31}x^{(i)} - a_{32}y^{(i)}\right]$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[a11*x+a12*y+a13*z=b1,a21*x+a22*y+a23*z=b2,a31*x+a32*y+a33*z=b3]$
print(“Given System of Equations is”)$
while EQ # [ ] do disp(pop(EQ))$
print(“Gauss-Seidel Iteration scheme for given system is”)$
x[i]:=(b1-a12*y[i-1]-a13*z[i-1])/a11;
y[i]:=(b2-a21*x[i]-a23*z[i-1])/a22;
z[i]:=(b3-a31*x[i]-a32*y[i])/a33;
x[0]:0$
y[0]:0$
z[0]:0$
i:1$
accuracy:0.00001$
print(“Initial Approximation is “,’x[0]=x[0],’y[0]=y[0],’z[0]=z[0])$
N:[[“Iteration No.”,”x”,”y”,”z”]]$
block(loop,N:push([i,x[i],y[i],z[i]],N),
   if abs(x[i]-x[i-1])<=accuracy and abs(y[i]-y[i-1])<=accuracy and
   abs(z[i]-z[i-1])<=accuracy then (table_form(reverse(N)),
      print(“Solution by Gauss-Seidel Method is x=”,x[i], “y=”,y[i],”z=”,z[i]))
else(i:i+1,go(loop)))$
```

Note: You may take **accuracy:0.001, 0.0001, 0.00001** to get approximate root correct to **2 decimal places, 3 decimal places, 4 decimal places** respectively.

Worked Examples:

Problem 1. Write a program to find the solution of given linear system of equations with initial guess by Gauss-Seidel Method performing 6 iterations.

$$10x + y + z = 12;\ 2x + 10y + z = 13;\ 2x + 2y + 10z = 14$$

$$x^{(0)} = y^{(0)} = z^{(0)} = 0$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[10·x+y+z=12,2·x+10·y+z=13,2·x+2·y+10·z=14]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Gauss-Seidel Iteration scheme for given system is")$
x[i]:=(12−y[i−1]−z[i−1])/10;
y[i]:=(13−2·x[i]−z[i−1])/10;
z[i]:=(14−2·x[i]−2·y[i])/10;
x[0]:0$
y[0]:0$
z[0]:0$
n:6$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
N:[["Iteration No.","x","y","z"]]$
for i:1 thru n do N:push([i,x[i],y[i],z[i]],N)$
table_form(reverse(N))$
print("Solution by Gauss-Seidel Method is x=",x[n], "y=",y[n],"z=",z[n])$
```

Output:

Given System of Equations is

$z+y+10\ x=12$

$z+10\ y+2\ x=13$

$10\ z+2\ y+2\ x=14$

Gauss−Seidel Iteration scheme for given system is

$$x_i:=\frac{12-y_{i-1}-z_{i-1}}{10}$$

$$y_i:=\frac{13-2\,x_i-z_{i-1}}{10}$$

$$z_i:=\frac{14-2\,x_i+(-2)\,y_i}{10}$$

Initial Approximation is $x_0=0\ \ y_0=0\ \ z_0=0$

Iteration No.	*x*	*y*	*z*
1	1.2	1.06	0.948
2	0.9992	1.00536	0.999088
3	0.9995552	1.00018	1.000053
4	0.9999767	0.9999994	1.000005
5	0.9999996	0.9999996	1.0
6	1.0	1.0	1.0

Solution by Gauss−Seidel Method is $x=\ 1.0\ \ y=\ 1.0\ \ z=\ 1.0$

Problem 2. Write a program to find the solution of given linear system of equations with initial guess by Gauss-Seidel Method performing 6 iterations.

$$5x - y + 3z = 10;\ 3x + 6y = 18;\ x + y + 5z = -10$$

$$x^{(0)} = 3, \qquad y^{(0)} = 0, \qquad z^{(0)} = -2$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[5·x−y+3·z=10,3·x+6·y=18,x+y+5·z=−10]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Gauss-Seidel Iteration scheme for given system is")$
x[i]:=(10+y[i−1]−3·z[i−1])/5;
y[i]:=(18−3·x[i])/6;
z[i]:=(−10−x[i]−y[i])/5;
x[0]:3$
y[0]:0$
z[0]:−2$
n:6$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
N:[["Iteration No.","x","y","z"]]$
for i:1 thru n do N:push([i,x[i],y[i],z[i]],N)$
table_form(reverse(N))$
print("Solution by Gauss-Seidel Method is x=",x[n], "y=",y[n],"z=",z[n])$
```

Output:

Given System of Equations is

$3\ z - y + 5\ x = 10$

$6\ y + 3\ x = 18$

$5\ z + y + x = -10$

Gauss−Seidel Iteration scheme for given system is

$$x_i := \frac{10 + y_{i-1} + (-3)\ z_{i-1}}{5}$$

$$y_i := \frac{18 - 3\ x_i}{6}$$

$$z_i := \frac{-10 - x_i - y_i}{5}$$

Initial Approximation is $x_0 = 3\ \ y_0 = 0\ \ z_0 = -2$

Iteration No.	x	y	z
1	3.2	1.4	−2.92
2	4.032	0.984	−3.0032
3	3.99872	1.00064	−2.999872
4	4.000051	0.9999744	−3.000005
5	3.999998	1.000001	−3.0
6	4.0	1.0	−3.0

Solution by Gauss−Seidel Method is x= 4.0 *y=* 1.0 *z=* −3.0

Problem 3. Write a program to find the solution of given linear system of equations with initial guess by Gauss-Seidel Method correct to 6 decimal places.

$$10x - y + 2z = 6, -x + 11y - z + 3t = 25, 2x - y + 10z - t = -11, 3y - z + 8t = 15$$

$$x^{(0)} = y^{(0)} = z^{(0)} = t^{(0)} = 0$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[10·x−y+2·z=6,−x+11·y−z+3·t=25,2·x−y+10·z−t=−11,3·y−z+8·t=15]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Gauss-Seidel Iteration scheme for given system is")$
x[i]:=(6+y[i−1]−2·z[i−1])/10;
y[i]:=(25+x[i]+z[i−1]−3·t[i−1])/11;
z[i]:=(−11−2·x[i]+y[i]+t[i−1])/10;
t[i]:=(15−3·y[i]+z[i])/8;
x[0]:0$
y[0]:0$
z[0]:0$
t[0]:0$
i:1$
accuracy:0.0000001$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0],'t[0]=t[0])$
N:[["Iteration No.","x","y","z","t"]]$
block(loop,N:push([i,x[i],y[i],z[i],t[i]],N),
  if abs(x[i]−x[i−1])<=accuracy and abs(y[i]−y[i−1])<=accuracy and
  abs(z[i]−z[i−1])<=accuracy and abs(t[i]−t[i−1])<=accuracy then (table_form(reverse(N)),
    print("Solution by Gauss-Seidel Method is x=",x[i], "y=",y[i],"z=",z[i], "t=",t[i]))
else(i:i+1,go(loop)))$
```

Output:

Given System of Equations is

$2z - y + 10x = 6$

$-z + 11y - x + 3t = 25$

$10z - y + 2x - t = -11$

$-z + 3y + 8t = 15$

Gauss−Seidel Iteration scheme for given system is

$$x_i := \frac{6 + y_{i-1} + (-2)z_{i-1}}{10}$$

$$y_i := \frac{25 + x_i + z_{i-1} + (-3)t_{i-1}}{11}$$

$$z_i := \frac{-11 - 2x_i + y_i + t_{i-1}}{10}$$

$$t_i := \frac{15 - 3y_i + z_i}{8}$$

Initial Approximation is $x_0 = 0 \;\; y_0 = 0 \;\; z_0 = 0 \;\; t_0 = 0$

Iteration No.	*x*	*y*	*z*	*t*
1	0.6	2.327273	−0.9872727	0.8788636
2	1.030182	2.036938	−1.014456	0.9843412
3	1.006585	2.003555	−1.002527	0.9983509
4	1.000861	2.000298	−1.000307	0.9998497
5	1.000091	2.000021	−1.000031	0.9999881
6	1.000008	2.000001	−1.000003	0.9999992
7	1.000001	2.0	−1.0	1.0
8	1.0	2.0	−1.0	1.0
9	1.0	2.0	−1.0	1.0

Solution by Gauss−Seidel Method is x= 1.0 *y=* 2.0 *z=* −1.0 *t=* 1.0

Problem 4. Write a program to find the solution of given linear system of equations with initial guess by Gauss-Jacobi Method performing 10 iterations. What is your observation about convergence of iteration?

$$2x - 6y + 8z = 24;\ 5x + 4y - 3z = 2;\ 3x + y + 2z = 16$$

$$x^{(0)} = 0, \quad y^{(0)} = 0, \quad z^{(0)} = 0$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[2·x−6·y+8·z=24,5·x+4·y−3·z=2,3·x+y+2·z=16]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("Gauss-Seidel Iteration scheme for given system is")$
x[i]:=(24+6·y[i−1]−8·z[i−1])/2;
y[i]:=(2−5·x[i]+3·z[i−1])/4;
z[i]:=(16−3·x[i]−y[i])/2;
x[0]:0$
y[0]:0$
z[0]:0$
n:5$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
N:[["Iteration No.","x","y","z"]]$
for i:1 thru n do N:push([i,x[i],y[i],z[i]],N)$
table_form(reverse(N))$
print("Solution by Gauss-Seidel Method is x=",x[n], "y=",y[n],"z=",z[n])$
```

Output:

Given System of Equations is

$8\,z - 6\,y + 2\,x = 24$

$-3\,z + 4\,y + 5\,x = 2$

$2\,z + y + 3\,x = 16$

Gauss−Seidel Iteration scheme for given system is

$$x_i := \frac{24 + 6\,y_{i-1} + (-8)\,z_{i-1}}{2}$$

$$y_i := \frac{2 - 5\,x_i + 3\,z_{i-1}}{4}$$

$$z_i := \frac{16 - 3\,x_i - y_i}{2}$$

Initial Approximation is $x_0 = 0 \quad y_0 = 0 \quad z_0 = 0$

Iteration No.	x	y	z
1	12	−14.5	−2.75
2	−20.5	24.0625	26.71875
3	−22.6875	48.89844	17.58203
4	88.36719	−96.77246	−76.16455
5	26.34082	−89.54944	13.26349

Solution by Gauss−Seidel Method is x= 26.34082 *y=* −89.54944 *z=* 13.26349

Observation: Clearly, Gauss-Seidel iteration is **not** convergent as the given system is **not** diagonally dominant

Exercise:

Write a program to find the solution of given linear system of equations with initial guess by Gauss-Seidel Method.

1. $\boldsymbol{5x - y + z = 10;\ x + 2y = 6;\ x + y + 5z = -1}$
   $\boldsymbol{x^{(0)} = 2,\ y^{(0)} = 3,\ z^{(0)} = 0}$ by performing 8 iterations.

   (Answer: $x = 2.555556, y = 1.722222, z = -1.055556$)

2. $\boldsymbol{27x + 6y - z = 85, 6x + 15y + 2z = 72, x + y + 54z = 110}$
   $\boldsymbol{x^{(0)} = 0,\ y^{(0)} = 0,\ z^{(0)} = 0}$ correct to 4 decimal places.

   (Answer: $x = 2.425476, y = 3.573016, z = 1.925954$)

3. $\boldsymbol{5x + 2y + z = 12;\ x + 4y + 2z = 15;\ x + 2y + 5z = 20}$
   $\boldsymbol{x^{(0)} = 0,\ y^{(0)} = 0,\ z^{(0)} = 0}$ correct to 4 decimal places.

   (Answer: $x = 1.0, y = 2.0, z = 3.0$)

4. $\boldsymbol{10x - 2y - z - t = 2}$
   $\boldsymbol{-2x + 10y - z - t = 14}$
   $\boldsymbol{-x - y + 10z - 2t = 25}$
   $\boldsymbol{-x - y - 2z + 10t = 1}$
   $\boldsymbol{x^{(0)} = 0,\ y^{(0)} = 0,\ z^{(0)} = 0, t^{(0)} = 0}$ correct to 6 decimal places.

   (Answer: $x = 1, y = 2, z = 3, t = 1$)

Program 7

Program solve system of algebraic equations using Successive Over Relaxation (SOR) Method.

Aim: To find the solution of given system of linear algebraic equations from initial guess and relaxation factor by SOR Method using Mathematics Softwares (FOSS).

Software: Maxima

Keys:

Key	Function
kill (all)	Unbinds all items on all infolists
float *(expr)*	Converts integers, rational numbers and bigfloats in *expr* to floating point numbers
numer:true	numer causes some mathematical functions (including exponentiation) with numerical arguments to be evaluated in floating point. Default value is false.
fpprintprec	This is an option variable to decide the number of digits to print when printing an ordinary float or bigfloat number. Default value is 16. Set any integer from 2 to 16.
:=	The function definition operator
define *(f(x_1, …, x_n), expr)*	Defines a function named *f* with arguments *x_1*, …, *x_n* and function body *expr*.
memoizing function *f*[*x_1*, ..., *x_n*] := *expr*	A memoizing function caches the result the first time it is called with a given argument, and returns the stored value, without recomputing it, when that same argument is given.
[a_1, a_2,…,a_m]	List of numbers/objects a_1, a_2,…,a_m.
if cond_1 *then* expr_1 *else* expr_0	evaluates to expr_1 *if* cond_1 evaluates to true, otherwise the expression evaluates to expr_0.
print (“*text*”, *expr*)$	Displays *text* within inverted commas and evaluates and displays *expr*
block ([v_1,…, v_m], expr_1, …, expr_n)	The function *block* allows to make the variables *v_1*, …, *v_m* to be local for a sequence of commands.
push *(item, list)*	*push* prepends the item *item* to the list *list* and returns a copy of the new list
pop (list)	pop removes and returns the first element from the list list.
reverse *(list)*	Reverses the order of the members of the *list* (not the members themselves)
table_form()	Displays a 2D list in a form that is more readable than the output from *Maxima*’s default output routine. The input is a list of one or more lists.
<=	less than or equal to
L[i]	Subscript operator for L_i
diff (expr, x)	Returns the first derivative of expr with respect to the variable x

Note:1. Press Shift+Enter for evaluation of commands and display of output.
2. Replace semicolon (;) by dollar ($) to suppress output of any input line and vice-versa.
3. Start each session with kill(all)$ or quit()$ to remove previously assigned values of all symbols

Definitions and Formulae:

Relaxation Method and Successive Over-Relaxation (SOR) Method:

Iteration scheme of Gauss-Seidel method for solving square linear system $\boldsymbol{AX = B}$ is given by:

$$\boldsymbol{X^{(k+1)} = D^{-1}\left[B - LX^{(k+1)} - UX^{(k)}\right]}$$

The element-wise formula of Gauss-Seidel method is

$$\boldsymbol{x_i}^{(k+1)} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_i^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_i^{(k)}\right], i = 1, 2, 3, \dots n$$

Relaxation method refers to a slightly modified version of Gauss-Seidel method. The modification is aimed at faster convergence. The iteration formula for Relaxation method is given by:

$$x_i^{(k+1)} = (1 - \omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_i^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_i^{(k)}\right], i = 1, 2, 3, \dots n$$

where the parameter $\boldsymbol{\omega} \in (\mathbf{0}, \mathbf{2})$ is called the relaxation parameter. Since this step is applied "successively" to each component during iteration process, the method is called successive relaxation method. When $\boldsymbol{\omega} = \mathbf{1}$, there is no relaxation and the method is same as the Gauss-Seidel method. When $\mathbf{0} < \boldsymbol{\omega} < \mathbf{1}$, the method is called **Successive Under-Relaxation Method** and when $\mathbf{1} < \boldsymbol{\omega} < \mathbf{2}$, the method is called **Successive Over-Relaxation (SOR) Method.** The choice of relaxation factor ω is not necessarily easy, and depends upon the properties of the coefficient matrix.

Program:

Program to find the solution of given linear system of equations

$$a_{11}x + a_{12}y + a_{13}z = b_1$$
$$a_{21}x + a_{22}y + a_{23}z = b_2$$
$$a_{31}x + a_{32}y + a_{33}x = b_3$$

with given relaxation factor and initial guess

$$x^{(0)} = y^{(0)} = z^{(0)} = 0, \omega = \omega_0$$

by SOR Method by performing n **iterations.**

Solving given system for diagonal entries, we get SOR iteration scheme as

$$x^{(i)} = (1-\omega)x^{(i-1)} + \frac{\omega}{a_{11}}\left[b_1 - a_{12}y^{(i-1)} - a_{13}z^{(i-1)}\right]$$
$$y^{(i)} = (1-\omega)y^{(i-1)} + \frac{\omega}{a_{22}}\left[b_2 - a_{21}x^{(i)} - a_{23}z^{(i-1)}\right]$$
$$z^{(i)} = (1-\omega)z^{(i-1)} + \frac{\omega}{a_{33}}\left[b_3 - a_{31}x^{(i)} - a_{32}y^{(i)}\right]$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[a11*x+a12*y+a13*z=b1,a21*x+a22*y+a23*z=b2,a31*x+a32*y+a33*z=b3]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("SOR Iteration scheme for given system is")$
x[i]:=(1-ω)*x[i-1]+ ω*(b1-a12*y[i-1]-a13*z[i-1])/a11;
y[i]:=(1-ω)*x[i-1]+ ω*(b2-a21*x[i-1]-a23*z[i-1])/a22;
z[i]:= (1-ω)*x[i-1]+ ω*(b3-a31*x[i-1]-a32*y[i-1])/a33;
x[0]:0$
y[0]:0$
z[0]:0$
ω:given relaxation parameter$
n:given number of iterations$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
print("Relaxation factor is ω =", ω)$
N:[["Iteration No.","x","y","z"]]$
for i:1 thru n do N:push([i,x[i],y[i],z[i]],N)$
table_form(reverse(N))$
print("Solution by Gauss-Jacobi Method is x=",x[n], "y=",y[n],"z=",z[n])$
```

Program:

Program to find the solution of given linear system of equations

$$a_{11}x + a_{12}y + a_{13}z = b_1$$
$$a_{21}x + a_{22}y + a_{23}z = b_2$$
$$a_{31}x + a_{32}y + a_{33}x = b_3$$

with given relaxation factor and initial guess

$$x^{(0)} = y^{(0)} = z^{(0)} = 0, \omega = \omega_0$$

by SOR Method by for **given accuracy**.

Solving given system for diagonal entries, we get SOR iteration scheme as

$$x^{(i)} = (1-\omega)x^{(i-1)} + \frac{\omega}{a_{11}}\left[b_1 - a_{12}y^{(i-1)} - a_{13}z^{(i-1)}\right]$$
$$y^{(i)} = (1-\omega)y^{(i-1)} + \frac{\omega}{a_{22}}\left[b_2 - a_{21}x^{(i)} - a_{23}z^{(i-1)}\right]$$
$$z^{(i)} = (1-\omega)z^{(i-1)} + \frac{\omega}{a_{33}}\left[b_3 - a_{31}x^{(i)} - a_{32}y^{(i)}\right]$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[a11*x+a12*y+a13*z=b1,a21*x+a22*y+a23*z=b2,a31*x+a32*y+a33*z=b3]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("SOR Iteration scheme for given system is")$
x[i]:=(1- ω)*x[i-1]+ ω*(b1-a12*y[i-1]-a13*z[i-1])/a11;
y[i]:=(1- ω)*x[i-1]+ ω*(b2-a21*x[i-1]-a23*z[i-1])/a22;
z[i]:= (1- ω)*x[i-1]+ ω*(b3-a31*x[i-1]-a32*y[i-1])/a33;
x[0]:0$
y[0]:0$
z[0]:0$
ω:given relaxation parameter$
i:1$
accuracy:0.00001$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
print("Relaxation factor is ω=", ω)$
N:[["Iteration No.","x","y","z"]]$
block(loop,N:push([i,x[i],y[i],z[i]],N),
   if abs(x[i]-x[i-1])<=accuracy and abs(y[i]-y[i-1])<=accuracy and
   abs(z[i]-z[i-1])<=accuracy then (table_form(reverse(N)),
      print("Solution by Gauss-Jacobi Method is x=",x[i], "y=",y[i],"z=",z[i]))
else(i:i+1,go(loop)))$
```

Note: You may take **accuracy:0.001, 0.0001, 0.00001** to get approximate root correct to **2 decimal places, 3 decimal places, 4 decimal places** respectively.

Worked Examples:

Problem 1. Write a program to find the solution of given linear system of equations with given relaxation parameter and initial guess by SOR Method performing 12 iterations.

$$2x+z=6;\ 2y+z=3;\ y+2z=4.5$$

$$x^{(0)}=y^{(0)}=z^{(0)}=0,\omega=1.25$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[2·x+z=6,2·y+z=3,y+2·z=4.5]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("SOR Iteration scheme for given system is")$
x[i]:=(1−ω)·x[i−1]+ω·(6−z[i−1])/2;
y[i]:=(1−ω)·y[i−1]+ω·(3−z[i−1])/2;
z[i]:=(1−ω)·z[i−1]+ω·(4.5−y[i])/2;
x[0]:0$
y[0]:0$
z[0]:0$
ω:1.25$
n:12$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
print("Relaxation factor is ω=",ω)$
N:[["Iteration No.","x","y","z"]]$
for i:1 thru n do N:push([i,x[i],y[i],z[i]],N)$
table_form(reverse(N))$
print("Solution by SOR Method is x=",x[n], "y=",y[n],"z=",z[n])$
```

Output:

Given System of Equations is

$z+2x=6$

$z+2y=3$

$2z+y=4.5$

SOR Iteration scheme for given system is

$$x_i:=(1-\omega)\,x_{i-1}+\frac{\omega\,(6-z_{i-1})}{2}$$

$$y_i:=(1-\omega)\,y_{i-1}+\frac{\omega\,(3-z_{i-1})}{2}$$

$$z_i:=(1-\omega)\,z_{i-1}+\frac{\omega\,(4.5-y_i)}{2}$$

Initial Approximation is $x_0=0\ \ y_0=0\ \ z_0=0$

Relaxation factor is $\omega=$ 1.25

Iteration No.	*x*	*y*	*z*
1	3.75	1.875	1.640625
2	1.787109	0.3808594	2.164307
3	1.950531	0.4270935	2.00449
4	2.009561	0.5154204	1.98924
5	2.004335	0.50287	2.000896
6	1.998356	0.4987223	2.000574
7	2.000052	0.4999604	1.999881
8	2.000061	0.5000842	1.999977
9	1.999999	0.4999933	2.00001
10	1.999994	0.4999955	2.0
11	2.000001	0.5000009	1.999999
12	2.0	0.5000002	2.0

Solution by SOR Method is $x=$ 2.0 $y=$ 0.5000002 $z=$ 2.0

Problem 2. Write a program to find the solution of given linear system of equations with given relaxation parameter and initial guess by SOR Method correct to 4 decimal places.

$$45x + 2y + 3z = 58;\ -3x + 22y + 2z = 47;\ 5x + y + 20z = 67$$

$$x^{(0)} = y^{(0)} = z^{(0)} = 0, \omega = 1.1$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[45·x+2·y+3·z=58,−3·x+22·y+2·z=47,5·x+y+20·z=67]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("SOR Iteration scheme for given system is")$
x[i]:=(1−ω)·x[i−1]+ω·(58−2·y[i−1]−3·z[i−1])/45;
y[i]:=(1−ω)·y[i−1]+ω·(47+3·x[i]−2·z[i−1])/22;
z[i]:=(1−ω)·z[i−1]+ω·(67−5·x[i]−y[i])/20;
x[0]:0$
y[0]:0$
z[0]:0$
i:1$
ω:1.1$
accuracy:0.00001$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
print("Relaxation factor is ω=",ω)$
N:[["Iteration No.","x","y","z","t"]]$
block(loop,N:push([i,x[i],y[i],z[i]],N),
  if abs(x[i]−x[i−1])<=accuracy and abs(y[i]−y[i−1])<=accuracy and
  abs(z[i]−z[i−1])<=accuracy then (table_form(reverse(N)),
    print("Solution by SOR Method is x=",x[i], "y=",y[i],"z=",z[i]))
else(i:i+1,go(loop)))$
```

Output:

Given System of Equations is

$3z+2y+45x=58$

$2z+22y-3x=47$

$20z+y+5x=67$

SOR Iteration scheme for given system is

$$x_i:=(1-\omega)\,x_{i-1}+\frac{\omega\,(58-2\,y_{i-1}+(-3)\,z_{i-1})}{45}$$

$$y_i:=(1-\omega)\,y_{i-1}+\frac{\omega\,(47+3\,x_i+(-2)\,z_{i-1})}{22}$$

$$z_i:=(1-\omega)\,z_{i-1}+\frac{\omega\,(67-5\,x_i-y_i)}{20}$$

Initial Approximation is $x_0=0\ \ y_0=0\ \ z_0=0$

Relaxation factor is ω= 1.1

Iteration No.	*x*	*y*	*z*	*t*
1	1.417778	2.562667	3.154164	
2	0.9194087	1.916228	3.011354	
3	1.011322	2.00894	2.995259	
4	0.9987784	1.999397	3.000843	
5	1.00009	1.999989	2.999892	
6	0.9999995	2.000012	3.00001	
7	0.9999987	1.999998	2.999999	
8	1.0	2.0	3.0	

Solution by SOR Method is x= 1.0 *y=* 2.0 *z=* 3.0

Problem 3. Write a program to find the solution of given linear system of equations with given relaxation parameter and initial guess by SOR Method correct to 4 decimal places.

$$4x - y + 2z = -2;\ -2x + 4y + 5z = -4;\ x + 2y + 5z = -5$$

$$x^{(0)} = 0, \quad y^{(0)} = 0, \quad z^{(0)} = 0, \quad \omega = 1.52$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[4·x−y+2·z=−2,−2·x+4·y+5·z=−4,x+2·y+5·z=−5]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("SOR Iteration scheme for given system is")$
x[i]:=(1−ω)·x[i−1]+ω·(−2+y[i−1]−2·z[i−1])/4;
y[i]:=(1−ω)·y[i−1]+ω·(−4+2·x[i]−5·z[i−1])/4;
z[i]:=(1−ω)·z[i−1]+ω·(−5−x[i]−2·y[i])/5;
x[0]:0$
y[0]:0$
z[0]:0$
i:1$
ω:1.52$
accuracy:0.00001$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0])$
print("Relaxation factor is ω=",ω)$
N:[["Iteration No.","x","y","z"]]$
block(loop,N:push([i,x[i],y[i],z[i]],N),
   if abs(x[i]−x[i−1])<=accuracy and abs(y[i]−y[i−1])<=accuracy and
   abs(z[i]−z[i−1])<=accuracy then (table_form(reverse(N)),
      print("Solution by SOR Method is x=",x[i], "y=",y[i],"z=",z[i]))
else(i:i+1,go(loop)))$
```

Output:

Given System of Equations is

$2z - y + 4x = -2$

$5z + 4y - 2x = -4$

$5z + 2y + x = -5$

SOR Iteration scheme for given system is

$$x_i := (1-\omega)\,x_{i-1} + \frac{\omega\,(-2 + y_{i-1} + (-2)\,z_{i-1})}{4}$$

$$y_i := (1-\omega)\,y_{i-1} + \frac{\omega\,(-4 + 2\,x_i + (-5)\,z_{i-1})}{4}$$

$$z_i := (1-\omega)\,z_{i-1} + \frac{\omega\,(-5 - x_i + (-2)\,y_i)}{5}$$

Initial Approximation is $x_0 = 0 \quad y_0 = 0 \quad z_0 = 0$

Relaxation factor is $\omega =$ 1.52

Iteration No.	*x*	*y*	*z*
1	−0.76	−2.0976	−0.0136192
2	−1.151537	−1.27854	−0.3854984
3	−0.354067	−0.3918033	−0.9736881
4	0.01523256	0.5453219	−1.349869
5	0.4652015	1.114736	−1.637249
6	0.6660042	1.517274	−1.793598
7	0.8333765	1.73222	−1.893865
8	0.9042256	1.864801	−1.943874
9	0.9557711	1.930049	−1.97321
10	0.9760575	1.967277	−1.986757
11	0.9899506	1.984216	−1.994235
12	0.9948464	1.993337	−1.99738
13	0.9981569	1.997086	−1.99903
14	0.9991143	1.999	−1.999627
15	0.9997969	1.999657	−1.999924
16	0.9999171	1.99997	−1.999996
17	1.000029	2.000031	−2.000029
18	1.000019	2.000054	−2.000024
19	1.000029	2.000038	−2.00002
20	1.000015	2.000029	−2.000012
21	1.000012	2.000016	−2.000008
22	1.000006	2.00001	−2.000004

Solution by SOR Method is $x = 1.000006$ $y = 2.00001$ $z = -2.000004$

Problem 4. Write a program to find the solution of given linear system of equations with given relaxation parameter and initial guess by SOR Method correct to 5 decimal places.

$$10x-2y-z-t=2;\ -2x+10y-z-t=14;\ -x-y+10z-2t=25;\ -x-y-2z+10t=1$$

$$x^{(0)}=y^{(0)}=z^{(0)}=t^{(0)}=0,\qquad \omega=1.1$$

Program:

```
kill(all)$
fpprintprec:7$
numer:true$
EQ:[10·x−2·y−z−t=2,−2·x+10·y−z−t=14,−x−y+10·z−2·t=25,−x−y−2·z+10·t=1]$
print("Given System of Equations is")$
while EQ # [ ] do disp(pop(EQ))$
print("SOR Iteration scheme for given system is")$
x[i]:=(1−ω)·x[i−1]+ω·(2+2·y[i−1]+z[i−1]+t[i−1])/10;
y[i]:=(1−ω)·y[i−1]+ω·(14+2·x[i]+z[i−1]+t[i−1])/10;
z[i]:=(1−ω)·z[i−1]+ω·(25+x[i]+y[i]+2·t[i−1])/10;
t[i]:=(1−ω)·t[i−1]+ω·(1+x[i]+y[i]+2·z[i])/10;
x[0]:0$
y[0]:0$
z[0]:0$
t[0]:0$
ω:1.1$
i:1$
accuracy:0.000001$
print("Initial Approximation is ",'x[0]=x[0],'y[0]=y[0],'z[0]=z[0],'t[0]=t[0])$
print("Relaxation factor is ω=",ω)$
N:[["Iteration No.","x","y","z","t"]]$
block(loop,N:push([i,x[i],y[i],z[i],t[i]],N),
  if abs(x[i]−x[i−1])<=accuracy and abs(y[i]−y[i−1])<=accuracy and
  abs(z[i]−z[i−1])<=accuracy and abs(t[i]−t[i−1])<=accuracy then (table_form(reverse(N)),
    print("Solution by SOR Method is x=",x[i], "y=",y[i],"z=",z[i], "t=",t[i]))
else(i:i+1,go(loop)))$
```

Output:

Given System of Equations is

$-z-2\,y+10\,x-t=2$

$-z+10\,y-2\,x-t=14$

$10\,z-y-x-2\,t=25$

$-2\,z-y-x+10\,t=1$

SOR Iteration scheme for given system is

$$x_i:=(1-\omega)\,x_{i-1}+\frac{\omega\,(2+2\,y_{i-1}+z_{i-1}+t_{i-1})}{10}$$

$$y_i:=(1-\omega)\,y_{i-1}+\frac{\omega\,(14+2\,x_i+z_{i-1}+t_{i-1})}{10}$$

$$z_i:=(1-\omega)\,z_{i-1}+\frac{\omega\,(25+x_i+y_i+2\,t_{i-1})}{10}$$

$$t_i:=(1-\omega)\,t_{i-1}+\frac{\omega\,(1+x_i+y_i+2\,z_i)}{10}$$

Initial Approximation is $x_0=0\ \ y_0=0\ \ z_0=0\ \ t_0=0$

Relaxation factor is $\omega=$ 1.1

Iteration No.	x	y	z	t
1	0.22	1.5884	2.948924	0.9576873
2	0.9771752	2.025866	2.996133	1.003715
3	1.007956	1.999147	3.001985	1.000847
4	0.9993283	2.000249	2.999941	0.9998559
5	1.0001	1.999975	2.999982	1.000019
6	0.9999846	1.999999	3.000004	0.9999973
7	1.000002	2.000001	2.999999	1.0
8	0.9999999	2.0	3.0	1.0
9	1.0	2.0	3.0	1.0

Solution by SOR Method is $x=$ 1.0 $y=$ 2.0 $z=$ 3.0 $t=$ 1.0

Exercise:

Write a program to find the solution of given linear system of equations with given relaxation parameter and initial guess by SOR Method.

1. $3x - y + z = -1$; $-x + 3y - z = 7$; $x - y + 3z = -7$

   $x^{(0)} = 0,\ y^{(0)} = 0,\ z^{(0)} = 0,\ \omega = 1.12$ by performing 11 iterations.

   (Answer: $x = 1.0, y = 2.0, z = -2.0$)

2. $4x + 3y = 24, 3x + 4y - z = 30, -y + 4z = -24$

   $x^{(0)} = 0,\ y^{(0)} = 0,\ z^{(0)} = 0, \omega = 1.3$ correct to 4 decimal places.

   (Answer: $x = 3.0, y = 4.0, z = -5.000001$)

3. $4x + 3y = 24$; $3x + 4y - z = 30$; $-y + 4z = -24$

   $x^{(0)} = 1,\ y^{(0)} = 1,\ z^{(0)} = 1$ correct to **5** decimal places.

   (Answer: $x = 3.0, y = 4.0, z = -5.0$)

4. $10x - y + 2z = 6$

   $-x + 11y - z + 3t = 25$

   $2x - y + 10z - t = -11$

   $3y - z + 8t = 15$

   $x^{(0)} = 0.5,\ y^{(0)} = 0.5,\ z^{(0)} = 0.5, t^{(0)} = 0.5, \omega = 1.1$ correct to 5 decimal places.

   (Answer: $x = 1.0, y = 2.0, z = -1.0, t = 1.0$)

Program 8

Program to evaluate integral using Simpson's 1/3rd and 3/8th Rules

Aim: To evaluate given definite integral using Simpson's 1/3rd and Simpson's 3/8th Rules using Mathematics Softwares (FOSS).

Software: Maxima

Keys:

Key	**Function**
kill (all)	Unbinds all items on all infolists
float *(expr)*	Converts integers, rational numbers and bigfloats in *expr* to floating point numbers
numer:true	numer causes some mathematical functions (including exponentiation) with numerical arguments to be evaluated in floating point. Default value is false.
fpprintprec	This is an option variable to decide the number of digits to print when printing an ordinary float or bigfloat number. Default value is 16. Set any integer from 2 to 16.
. (dot)	The operator . represents noncommutative multiplication and scalar product
:=	The function definition operator
integrate (expr, x, a, b)	Attempts to symbolically compute the integral of expr with respect to x, with limits of integration a and b
'	The single quote operator ' prevents evaluation.
makelist *(expr, i, i_0, i_max)*	Returns the list of elements obtained when ev *(expr, i=j)* is applied to the elements *j* of the sequence: *i_0, i_0* + *1, i_0* + 2, ..., with $\lvert j \rvert \le \lvert i\_max \rvert$.
define *(f(x_1, …, x_n), expr)*	Defines a function named *f* with arguments *x_1*, …, *x_n* and function body *expr*.
memoizing function *f*[*x_1*, ..., *x_n*] := *expr*	A memoizing function caches the result the first time it is called with a given argument, and returns the stored value, without recomputing it, when that same argument is given.
$[a_1, a_2, \ldots, a_m]$	List of numbers/objects $a_1, a_2, \ldots, a_m$.
if cond_1 *then* expr_1 *else* expr_0	evaluates to expr_1 *if* cond_1 evaluates to true, otherwise the expression evaluates to expr_0.
print ("*text*", *expr*)$	Displays *text* within inverted commas and evaluates and displays *expr*
block ([v_1, …, v_m], expr_1, …, expr_n)	The function *block* allows to make the variables *v_1*, …, *v_m* to be local for a sequence of commands.
push *(item, list)*	*push* prepends the item *item* to the list *list* and returns a copy of the new list
reverse *(list)*	Reverses the order of the members of the *list* (not the members themselves)
table_form()	Displays a 2D list in a form that is more readable than the output from *Maxima*'s default output routine. The input is a list of one or more lists.

Note:1. Press Shift+Enter for evaluation of commands and display of output.
2. Replace semicolon (;) by dollar ($) to suppress output of any input line and vice-versa.
3. Start each session with kill(all)$ or quit()$ to remove previously assigned values of all symbols

Definitions and Formulae:

Newton-Cote's Quadrature Formula for numerical integration: Numerical integration is the process of obtaining approximate value of the definite integral:

$$I = \int_a^b y dx = \int_a^b f(x) dx$$

without actually integrating the function but only using the values of $y = f(x)$ at some points of x equally spaced over $[a, b]$. Numerical integration is very useful when evaluation of integral is not easy or not possible. Newton-Cote's quadrature formula of order n also called General quadrature formula used for numerical integration of $y = f(x)$ from $n + 1$ data points $(x_i, y_i), i = 0,1,2, \dots, n$ is given by:

$$I = \int_a^b y dx = \int_a^b f(x) dx = w_0 y_0 + w_1 y_1 + w_2 y_2 + \cdots + w_n y_n$$

where $w_i, i = 0,2, \dots, n$ are called weights, which depend only on abscissa, $x_0 = a, x_n = b, h = (x_i - x_{i-1}) = \frac{(b-a)}{n}$. Approximating $f(x)$ by Newton's forward interpolation polynomial of order n or Lagrange interpolation polynomial of order n, we get different weights for different n, giving different Quadrature Rules. Quadrature rules and corresponding weights for some values of n are listed in the below table:

n	Quadrature rule	Weights	Formula
$n = 1$	**Trapezoidal Rule**	$\frac{h}{2}[1, 1]$	$\int_{x_0}^{x_1} y dx = \frac{h}{2}\{y_0 + y_1\}$
$n = 2$	**Simpson's 1/3rd Rule**	$\frac{h}{3}[1, 4, 1]$	$\int_{x_0}^{x_2} y dx = \frac{h}{3}\{y_0 + 4y_1 + y_2\}$
$n = 3$	**Simpson's 3/8th Rule**	$\frac{3h}{8}[1, 3, 3, 1]$	$\int_{x_0}^{x_3} y dx = \frac{3h}{8}\{y_0 + 3y_1 + 3y_2 + y_3\}$
$n = 6$	**Weddle's Rule**	$\frac{3h}{10}[1, 5, 1, 6, 1, 5, 1]$	$\int_{x_0}^{x_3} y dx = \frac{3h}{10}\{y_0 + 5y_1 + y_2 + 6y_3 + y_4 + 5y_5 + y_6\}$

Composite Quadrature Rules for above Rules are obtained by adding quadrature formulas applied for two or more consecutive intervals of equal lengths. Composite Simpson's 1/3rd rule and Simpson's 3/8th rules are given below:

Composite Simpson's 1/3rd Rule (Number of subintervals should be multiple of 2)

Number of subintervals	Weights	Formula
2 (2+1=3 points)	$\frac{h}{3}[1,4,1]$	$\int_{x_0}^{x_2} ydx = \frac{h}{3}\{y_0 + 4y_1 + y_2\}$
4 (4+1=5 points)	$\frac{h}{3}[1,4,2,4,1]$	$\int_{x_0}^{x_4} ydx = \frac{h}{3}\{y_0 + 4y_1 + 2y_2 + 4y_3 + y_4\}$
6 (6+1=7 points)	$\frac{h}{3}[1,4,2,4,2,4,1]$	$\int_{x_0}^{x_6} ydx = \frac{h}{3}\{y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + 4y_5 + y_6\}$
8 (8+1=9 points)	$\frac{h}{3}[1,4,2,4,2,4,2,4,1]$	$\int_{x_0}^{x_8} ydx = \frac{h}{3}\{y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + 4y_5 + 2y_6 + 4y_7 + y_8\}$
10 (10+1=11 points)	$\frac{h}{3}[1,4,2,4,2,4,2,4,2,4,1]$	$\int_{x_0}^{x_8} ydx = \frac{h}{3}\{y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + 4y_5 + 2y_6 + 4y_7 + 2y_8 + 4y_9 + y_{10}\}$

Composite Simpson's 3/8th Rule (Number of sub-intervals should be multiple of 3)

Number of subintervals	Weights	Formula
3 (3+1=4 points)	$\frac{3h}{8}[1,3,3,1]$	$\int_{x_0}^{x_3} ydx = \frac{3h}{8}\{y_0 + 3y_1 + 3y_2 + y_3\}$
6 (6+1=7 points)	$\frac{3h}{8}[1,3,3,2,3,3,1]$	$\int_{x_0}^{x_6} ydx = \frac{3h}{8}\{y_0 + 3y_1 + 3y_2 + 2y_3 + 3y_4 + 3y_5 + y_6\}$
9 (9+1=10 points)	$\frac{3h}{8}[1,3,3,2,3,3,2,3,3,1]$	$\int_{x_0}^{x_9} ydx = \frac{3h}{8}\{y_0 + 3y_1 + 3y_2 + 2y_3 + 3y_4 + 3y_5 + 2y_6 + 3y_7 + 3y_8 + y_9\}$
12 (12+1=13 points)	$\frac{3h}{8}[1,3,3,2,3,3,2,3,3,2,3,3,1]$	$\int_{x_0}^{x_{12}} ydx = \frac{3h}{8}\{y_0 + 3y_1 + 3y_2 + 2y_3 + 3y_4 + 3y_5 + 2y_6 + 3y_7 + 3y_8 + 2y_9 + 3y_{10} + 3y_{11} + y_{12}\}$

Program: (Simpson's 1/3rd Rule)

Program to evaluate $\int_a^b f(x)dx$ from given data points by Simpson's 1/3rd rule.

Note that $f(x)$ may be unknown or known but data is given.

x	x_0	x_1	x_2	x_3	x_4	x_5	x_6
y	y_0	y_1	y_2	y_3	y_4	y_5	y_6

Program:

```
kill(all)$
fpprintprec:5$
X:[given values of x separated by comma]$
Y:[given values of y separated by comma]$
a:first(X)$
b:last(X)$
h:abs(first(X)-second(X))$
I:h/3*[1,4,2,4,2,4,1].Y$
print("Given Data Table is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("By Simpson's 1/3 rd rule")$
'integrate(f(x),x,a,b)=I;
```

Program to evaluate $\int_a^b f(x)dx$ from given $f(x)$ by Simpson's 1/3rd rule.

Program:

```
kill(all)$
fpprintprec:6$
f(x):=given function$
a:lower limit of integral$
b:upper limit of integral$
n:6$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/3*[1,4,2,4,2,4,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 1/3 rd rule")$
'integrate(f(x),x,a,b)=I;
```

Note: Use weight vector of appropriate length. In the above illustration weight vector of length 7 is used for 7 data points (i.e. for n=6).

Program: (Simpson’s 3/8$^{\text{th}}$ Rule)

Program to evaluate $\int_a^b f(x)dx$ from given data points by Simpson’s 3/8$^{\text{th}}$ rule.

Note that $f(x)$ may be unknown or known but data is given.

x	x_0	x_1	x_2	x_3	x_4	x_5	x_6
y	y_0	y_1	y_2	y_3	y_4	y_5	y_6

Program:

```
kill(all)$
fpprintprec:5$
X:[given values of x separated by comma]$
Y:[given values of y separated by comma]$
a:first(X)$
b:last(X)$
h:abs(first(X)-second(X))$
I:3*h/8*[1,3,3,2,3,3,1].Y$
print("Given Data Table is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("By Simpson's 3/8 th rule")$
'integrate(f(x),x,a,b)=I;
```

Program to evaluate $\int_a^b f(x)dx$ from given $f(x)$ by Simpson’s 3/8$^{\text{th}}$ rule.

Program:

```
kill(all)$
fpprintprec:6$
f(x):=given function$
a:lower limit of integral$
b:upper limit of integral$
n:6$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:3*h/8*[1,3,3,2,3,3,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 3/8 th rule")$
'integrate(f(x),x,a,b)=I;
```

Note: Use weight vector of appropriate length. In the above illustration weight vector of length 7 is used for 7 data points (i.e. for n=6).

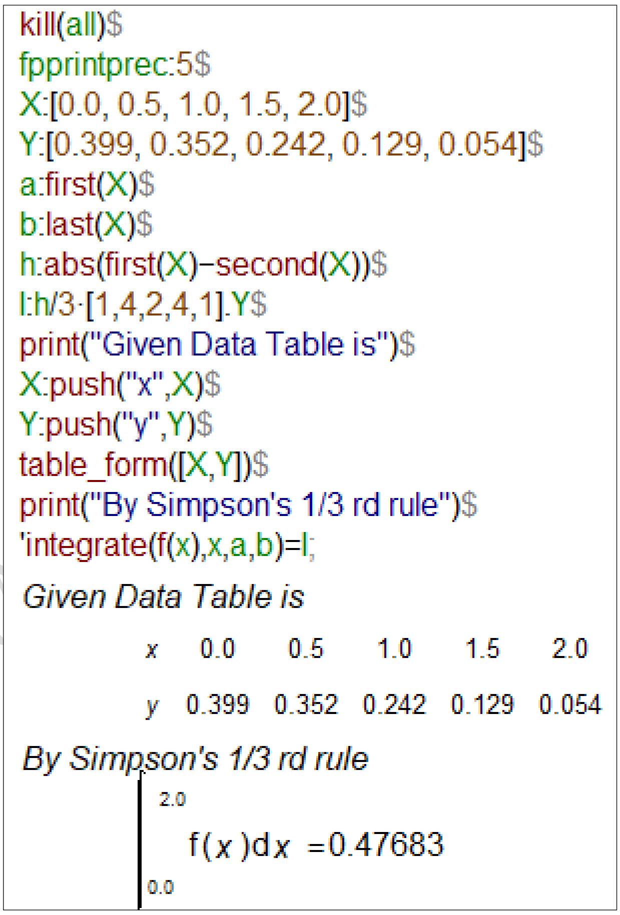
Worked Examples: (Simpson's 1/3rd Rule)

Problem 1. Write a program to evaluate the integral $\int_0^2 f(x)dx$ from the given data points by Simpson's 1/3rd rule.

x	0	0.5	1.0	1.5	2.0
y	0.399	0.352	0.242	0.129	0.054

Program:

```
kill(all)$
fpprintprec:5$
X:[0.0, 0.5, 1.0, 1.5, 2.0]$
Y:[0.399, 0.352, 0.242, 0.129, 0.054]$
a:first(X)$
b:last(X)$
h:abs(first(X)-second(X))$
I:h/3*[1,4,2,4,1].Y$
print("Given Data Table is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("By Simpson's 1/3 rd rule")$
'integrate(f(x),x,a,b)=I;
```



```
kill(all)$
fpprintprec:5$
X:[0.0, 0.5, 1.0, 1.5, 2.0]$
Y:[0.399, 0.352, 0.242, 0.129, 0.054]$
a:first(X)$
b:last(X)$
h:abs(first(X)−second(X))$
I:h/3·[1,4,2,4,1].Y$
print("Given Data Table is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("By Simpson's 1/3 rd rule")$
'integrate(f(x),x,a,b)=I;
```

Given Data Table is

x	0.0	0.5	1.0	1.5	2.0
y	0.399	0.352	0.242	0.129	0.054

By Simpson's 1/3 rd rule

$$\int_{0.0}^{2.0} f(x)dx = 0.47683$$

Output:

Given Data Table is

x	0.0	0.5	1.0	1.5	2.0
y	0.399	0.352	0.242	0.129	0.054

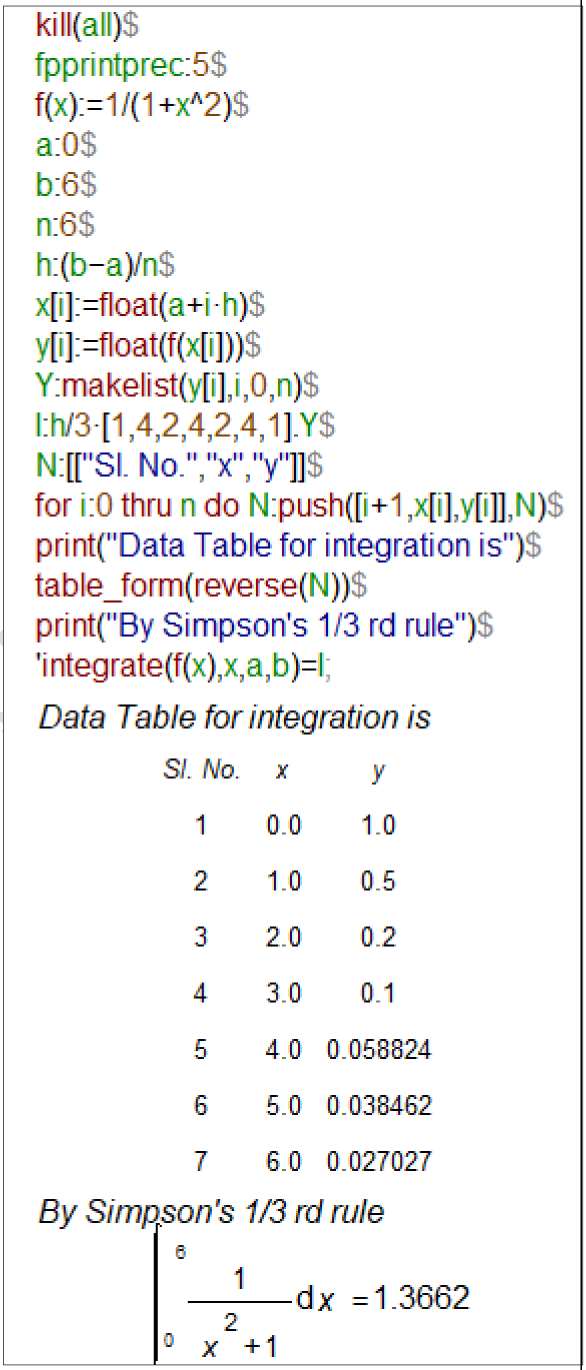
By Simpson's 1/3 rd rule

$$\int_{0.0}^{2.0} f(x)\,dx = 0.47683$$

Problem 2. Write a program to evaluate the integral $\int_0^6 \frac{dx}{1+x^2}$ by Simpson's 1/3rd rule by taking 6 subintervals.

Program:

```
kill(all)$
fpprintprec:5$
f(x):=1/(1+x^2)$
a:0$
b:6$
n:6$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/3*[1,4,2,4,2,4,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 1/3 rd rule")$
'integrate(f(x),x,a,b)=I;
```



```
kill(all)$
fpprintprec:5$
f(x):=1/(1+x^2)$
a:0$
b:6$
n:6$
h:(b−a)/n$
x[i]:=float(a+i·h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/3·[1,4,2,4,2,4,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 1/3 rd rule")$
'integrate(f(x),x,a,b)=I;
```

Data Table for integration is

Sl. No.	x	y
1	0.0	1.0
2	1.0	0.5
3	2.0	0.2
4	3.0	0.1
5	4.0	0.058824
6	5.0	0.038462
7	6.0	0.027027

By Simpson's 1/3 rd rule

$$\int_0^6 \frac{1}{x^2+1} dx = 1.3662$$

Output:

Data Table for integration is

Sl. No.	x	y
1	0.0	1.0
2	1.0	0.5
3	2.0	0.2
4	3.0	0.1
5	4.0	0.058824
6	5.0	0.038462
7	6.0	0.027027

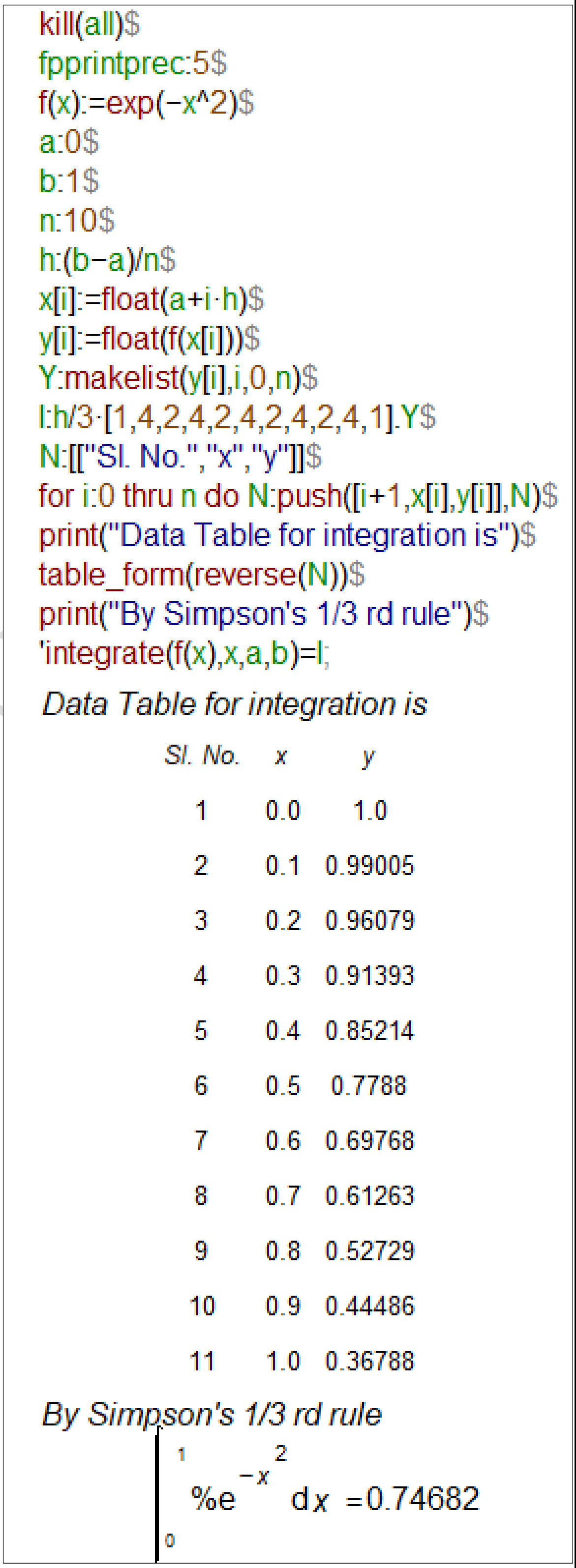
By Simpson's 1/3 rd rule

$$\int_0^6 \frac{1}{x^2+1} dx = 1.3662$$

Problem 3. Write a program to evaluate the integral $\int_0^1 e^{-x^2}dx$ by Simpson's 1/3rd rule by taking 10 subintervals.

Program:

```
kill(all)$
fpprintprec:5$
f(x):=exp(-x^2)$
a:0$
b:1$
n:10$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/3*[1,4,2,4,2,4,2,4,2,4,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 1/3 rd rule")$
'integrate(f(x),x,a,b)=I;
```



```
kill(all)$
fpprintprec:5$
f(x):=exp(−x^2)$
a:0$
b:1$
n:10$
h:(b−a)/n$
x[i]:=float(a+i·h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/3·[1,4,2,4,2,4,2,4,2,4,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 1/3 rd rule")$
'integrate(f(x),x,a,b)=I;
```

Data Table for integration is

Sl. No.	x	y
1	0.0	1.0
2	0.1	0.99005
3	0.2	0.96079
4	0.3	0.91393
5	0.4	0.85214
6	0.5	0.7788
7	0.6	0.69768
8	0.7	0.61263
9	0.8	0.52729
10	0.9	0.44486
11	1.0	0.36788

By Simpson's 1/3 rd rule

$$\int_0^1 \%e^{-x^2}dx = 0.74682$$

Output:

Data Table for integration is

Sl. No.	x	y
1	0.0	1.0
2	0.1	0.99005
3	0.2	0.96079
4	0.3	0.91393
5	0.4	0.85214
6	0.5	0.7788
7	0.6	0.69768
8	0.7	0.61263
9	0.8	0.52729
10	0.9	0.44486
11	1.0	0.36788

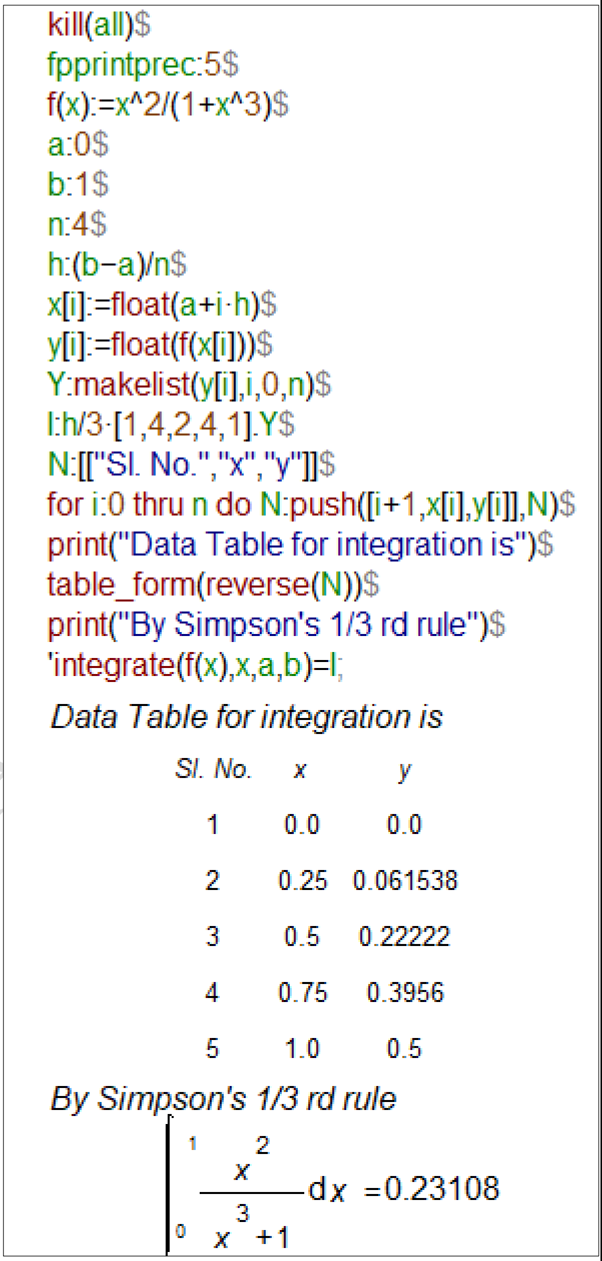
By Simpson's 1/3 rd rule

$$\int_0^1 \%e^{-x^2}dx = 0.74682$$

Problem 4. Write a program to evaluate the integral $\int_0^1 \frac{x^2}{x^3+1} dx$ by Simpson's 1/3rd rule by taking 5 ordinates.

Program:

```
kill(all)$
fpprintprec:5$
f(x):=x^2/(1+x^3)$
a:0$
b:1$
n:4$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/3*[1,4,2,4,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 1/3 rd rule")$
'integrate(f(x),x,a,b)=I;
```



```
kill(all)$
fpprintprec:5$
f(x):=x^2/(1+x^3)$
a:0$
b:1$
n:4$
h:(b−a)/n$
x[i]:=float(a+i·h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/3·[1,4,2,4,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 1/3 rd rule")$
'integrate(f(x),x,a,b)=I;
```

Data Table for integration is

Sl. No.	x	y
1	0.0	0.0
2	0.25	0.061538
3	0.5	0.22222
4	0.75	0.3956
5	1.0	0.5

By Simpson's 1/3 rd rule

$$\int_0^1 \frac{x^2}{x^3+1} dx = 0.23108$$

Output:

Data Table for integration is

$Sl. No.$	x	y
1	0.0	0.0
2	0.25	0.061538
3	0.5	0.22222
4	0.75	0.3956
5	1.0	0.5

By Simpson's 1/3 rd rule

$$\int_0^1 \frac{x^2}{x^3+1} dx = 0.23108$$

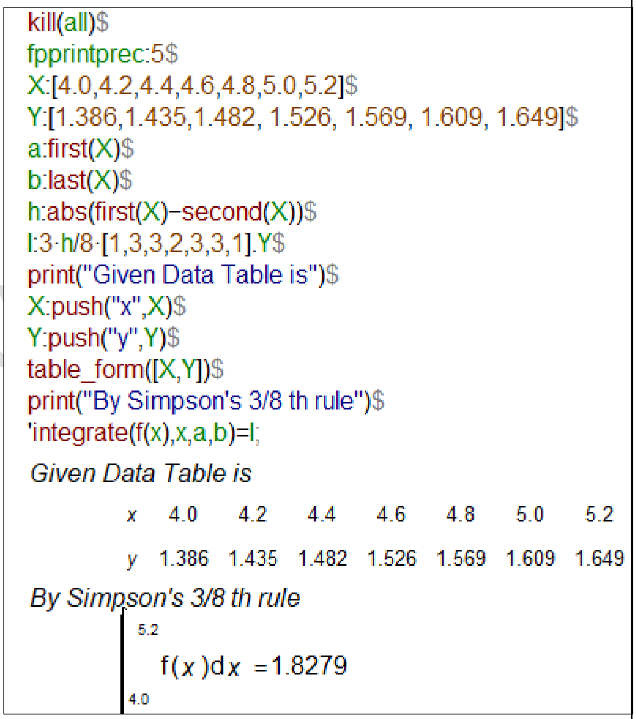
Worked Examples: (Simpson's 3/8th Rule)

Problem 5. Write a program to evaluate the integral $\int_{4.0}^{5.2} f(x)\,dx$ from the given data points by Simpson's 3/8th rule.

x	4.0	4.2	4.4	4.6	4.8	5.0	5.2
y	1.386	1.435	1.482	1.526	1.569	1.609	1.649

Program:

```
kill(all)$
fpprintprec:5$
X:[4.0,4.2,4.4,4.6,4.8,5.0,5.2]$
Y:[1.386,1.435,1.482, 1.526, 1.569, 1.609, 1.649]$
a:first(X)$
b:last(X)$
h:abs(first(X)-second(X))$
I:3*h/8*[1,3,3,2,3,3,1].Y$
print("Given Data Table is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("By Simpson's 3/8 th rule")$
'integrate(f(x),x,a,b)=I;
```



Output:

Given Data Table is

x	4.0	4.2	4.4	4.6	4.8	5.0	5.2
y	1.386	1.435	1.482	1.526	1.569	1.609	1.649

By Simpson's 3/8 th rule

$$\int_{4.0}^{5.2} f(x)\,dx = 1.8279$$

Problem 6. Write a program to evaluate the integral $\int_0^1 \frac{dx}{1+x^2}$ by Simpson's 3/8th rule by taking 6 subintervals.

Program:

```
kill(all)$
fpprintprec:5$
f(x):=1/(1+x^2)$
a:0$
b:1$
n:6$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:3*h/8*[1,3,3,2,3,3,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 3/8 th rule")$
'integrate(f(x),x,a,b)=I;
```


```
kill(all)$
fpprintprec:5$
f(x):=1/(1+x^2)$
a:0$
b:1$
n:6$
h:(b−a)/n$
x[i]:=float(a+i·h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:3·h/8·[1,3,3,2,3,3,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 3/8 th rule")$
'integrate(f(x),x,a,b)=I;
```

Data Table for integration is

Sl. No.	x	y
1	0.0	1.0
2	0.16667	0.97297
3	0.33333	0.9
4	0.5	0.8
5	0.66667	0.69231
6	0.83333	0.59016
7	1.0	0.5

By Simpson's 3/8 th rule

$$\int_0^1 \frac{1}{x^2+1}\,dx = 0.7854$$

Output:

Data Table for integration is

Sl. No.	x	y
1	0.0	1.0
2	0.16667	0.97297
3	0.33333	0.9
4	0.5	0.8
5	0.66667	0.69231
6	0.83333	0.59016
7	1.0	0.5

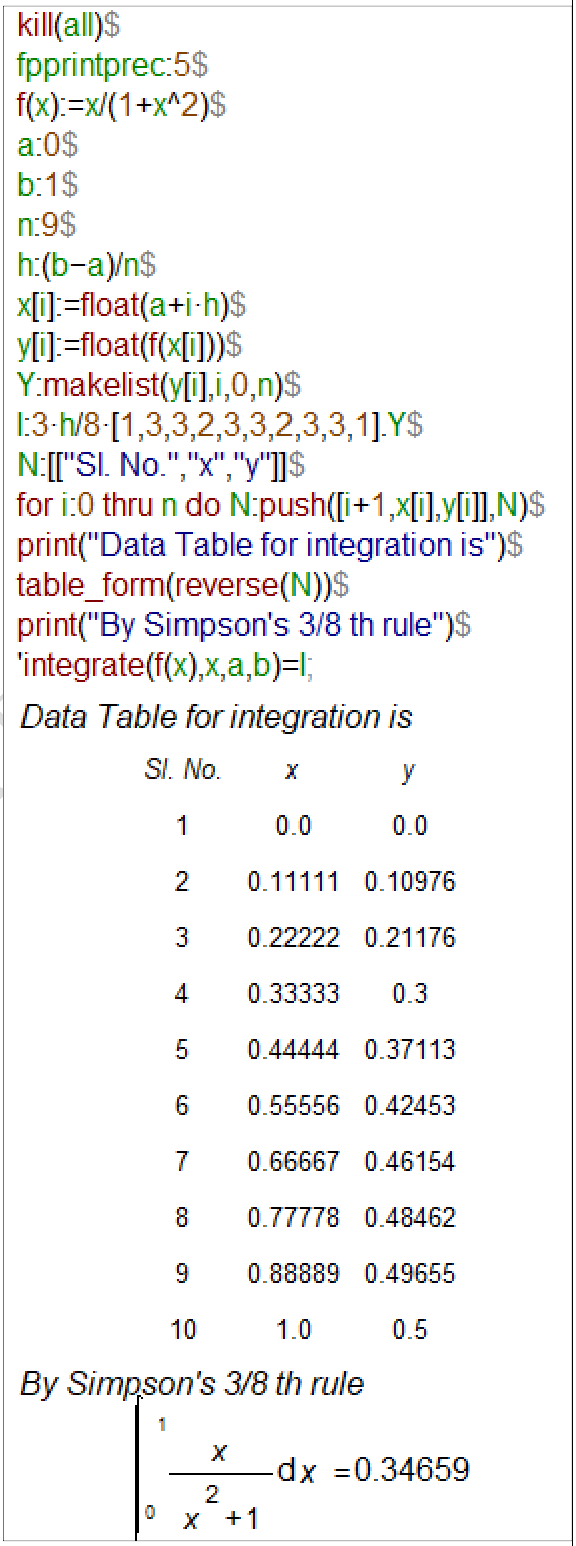
By Simpson's 3/8 th rule

$$\int_0^1 \frac{1}{x^2+1}\,dx = 0.7854$$

Problem 7. Write a program to evaluate the integral $\int_0^1 \frac{x\,dx}{1+x^2}$ by Simpson's $3/8^{th}$ rule by taking 9 subintervals.

Program:

```
kill(all)$
fpprintprec:5$
f(x):=x/(1+x^2)$
a:0$
b:1$
n:9$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:3*h/8*[1,3,3,2,3,3,2,3,3,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 3/8 th rule")$
'integrate(f(x),x,a,b)=I;
```



```
kill(all)$
fpprintprec:5$
f(x):=x/(1+x^2)$
a:0$
b:1$
n:9$
h:(b−a)/n$
x[i]:=float(a+i·h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:3·h/8·[1,3,3,2,3,3,2,3,3,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 3/8 th rule")$
'integrate(f(x),x,a,b)=I;
```

Data Table for integration is

Sl. No.	x	y
1	0.0	0.0
2	0.11111	0.10976
3	0.22222	0.21176
4	0.33333	0.3
5	0.44444	0.37113
6	0.55556	0.42453
7	0.66667	0.46154
8	0.77778	0.48462
9	0.88889	0.49655
10	1.0	0.5

By Simpson's 3/8 th rule

$$\int_0^1 \frac{x}{x^2+1}dx = 0.34659$$

Output:

$Data\ Table\ for\ integration\ is$

$Sl.No.$	x	y
1	0.0	0.0
2	0.11111	0.10976
3	0.22222	0.21176
4	0.33333	0.3
5	0.44444	0.37113
6	0.55556	0.42453
7	0.66667	0.46154
8	0.77778	0.48462
9	0.88889	0.49655
10	1.0	0.5

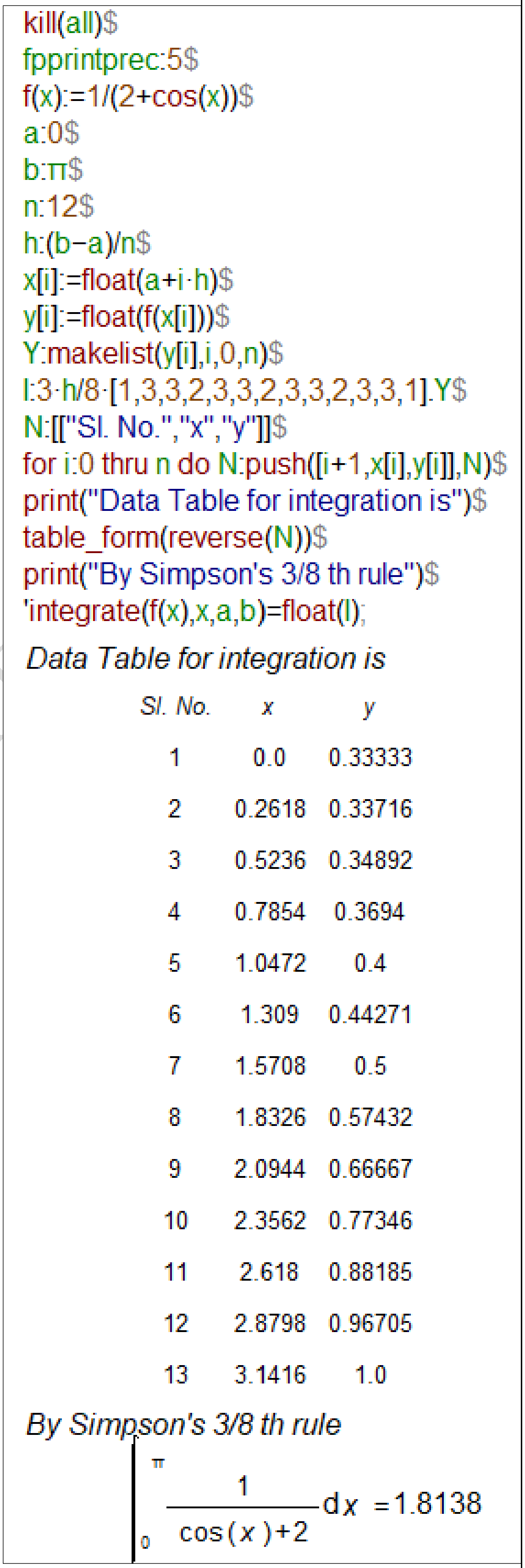
$By\ Simpson's\ 3/8\ th\ rule$

$$\int_0^1 \frac{x}{x^2+1}dx = 0.34659$$

Problem 8. Write a program to evaluate the integral $\int_0^{\pi} \frac{dx}{2+\cos(x)}$ by Simpson's 3/8th rule by taking 12 subintervals.

Program:

```
kill(all)$
fpprintprec:5$
f(x):=1/(2+cos(x))$
a:0$
b:π$
n:12$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:3*h/8*[1,3,3,2,3,3,2,3,3,2,3,3,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Simpson's 3/8 th rule")$
'integrate(f(x),x,a,b)=float(I);
```



Output:

Data Table for integration is

Sl. No.	x	y
1	0.0	0.33333
2	0.2618	0.33716
3	0.5236	0.34892
4	0.7854	0.3694
5	1.0472	0.4
6	1.309	0.44271
7	1.5708	0.5
8	1.8326	0.57432
9	2.0944	0.66667
10	2.3562	0.77346
11	2.618	0.88185
12	2.8798	0.96705
13	3.1416	1.0

By Simpson's 3/8 th rule

$$\int_0^{\pi} \frac{1}{\cos(x)+2}\,dx = 1.8138$$

Exercise:

I. Write a program to evaluate given definite integrals by Simpson's 1/3rd rule.

1. Evaluate $\int_{0.2}^{1.4} f(x)dx$ from given data points

x	0.2	0.4	0.6	0.8	1.0	1.2	1.4
y	0.199	0.389	0.565	0.717	0.841	0.932	0.985

(Answer: 0.80987)

2. Evaluate $\int_0^1 \frac{1}{(1+x)^2}dx$ taking 8 sub-intervals

(Answer: 0.50003)

3. Evaluate $\int_1^2 \frac{1}{\sqrt{3+2x-x^2}}dx$ taking 6 sub-intervals

(Answer: 0.5236)

4. Evaluate $\int_0^{\frac{\pi}{2}} e^{\sin(x)}dx$ taking 4 sub-intervals

(Answer: 3.1044)

II. Write a program to evaluate given definite integrals by Simpson's 3/8th rule.

1. Evaluate $\int_3^6 f(x)dx$ from given data points

x	3.0	3.5	4.0	4.5	5.0	5.5	6.0
y	0.4771	0.5440	0.6020	0.6532	0.6996	0.7404	0.7782

(Answer: 1.9349)

2. Evaluate $\int_0^1 \frac{1}{(1+x)^2}dx$ taking 9 sub-intervals

(Answer: 0.50004)

3. Evaluate $\int_0^{\frac{\pi}{2}} \sqrt{\sin(x)}\ dx$ taking 6 sub-intervals

(Answer: 1.1849)

4. Evaluate $\int_0^{\frac{\pi}{2}} \frac{\sqrt{\sin(x)}}{\sqrt{\sin(x)}+\sqrt{\cos(x)}}dx$ taking 3 sub-intervals

(Answer: 0.7854)

Program 9

Program to evaluate integral using Trapezoidal and Weddle Rules.

Aim: To evaluate given integral using Trapezoidal and Weddle's Rules using Mathematics Softwares (FOSS).

Software: Maxima

Keys:

Key	Function
kill (all)	Unbinds all items on all infolists
float *(expr)*	Converts integers, rational numbers and bigfloats in *expr* to floating point numbers
numer:true	numer causes some mathematical functions (including exponentiation) with numerical arguments to be evaluated in floating point. Default value is false.
fpprintprec	This is an option variable to decide the number of digits to print when printing an ordinary float or bigfloat number. Default value is 16. Set any integer from 2 to 16.
:=	The function definition operator
. (dot)	The operator . represents noncommutative multiplication and scalar product
integrate (expr, x, a, b)	Attempts to symbolically compute the integral of expr with respect to x, with limits of integration a and b
'	The single quote operator ' prevents evaluation.
makelist *(expr, i, i_0, i_max)*	Returns the list of elements obtained when ev (*expr, i=j*) is applied to the elements *j* of the sequence: *i_0, i_0* + *1, i_0* + 2, ..., with |*j*| ≤ |*i_max*|.
define *(f(x_1, …, x_n), expr)*	Defines a function named *f* with arguments *x_1*, …, *x_n* and function body *expr*.
memoizing function *f*[*x_1*, ..., *x_n*] := *expr*	A memoizing function caches the result the first time it is called with a given argument, and returns the stored value, without recomputing it, when that same argument is given.
$[a_1, a_2,\dots,a_m]$	List of numbers/objects $a_1, a_2,\dots,a_m$.
if cond_1 *then* expr_1 *else* expr_0	evaluates to expr_1 *if* cond_1 evaluates to true, otherwise the expression evaluates to expr_0.
print (“*text*”, *expr*)$	Displays *text* within inverted commas and evaluates and displays *expr*
block ([v_1, …, v_m], expr_1, …, expr_n)	The function *block* allows to make the variables *v_1*, …, *v_m* to be local for a sequence of commands.
push *(item, list)*	*push* prepends the item *item* to the list *list* and returns a copy of the new list
reverse *(list)*	Reverses the order of the members of the *list* (not the members themselves)
table_form()	Displays a 2D list in a form that is more readable than the output from *Maxima*'s default output routine. The input is a list of one or more lists.

Note:1. Press Shift+Enter for evaluation of commands and display of output.
2. Replace semicolon (;) by dollar ($) to suppress output of any input line and vice-versa.
3. Start each session with kill(all)$ or quit()$ to remove previously assigned values of all symbols

Definitions and Formulae:

Newton-Cote's Quadrature Formula for numerical integration: Numerical integration is the process of obtaining approximate value of the definite integral:

$$I = \int_a^b y dx = \int_a^b f(x) dx$$

without actually integrating the function but only using the values of $y = f(x)$ at some points of x equally spaced over $[a, b]$. Numerical integration is very useful when evaluation of integral is not easy or not possible. Newton-Cote's quadrature formula of order n also called General quadrature formula used for numerical integration of $y = f(x)$ from $n + 1$ data points $(x_i, y_i), i = 0,1,2, \dots, n$ is given by:

$$I = \int_a^b y dx = \int_a^b f(x) dx = w_0 y_0 + w_1 y_1 + w_2 y_2 + \cdots + w_n y_n$$

where $w_i, i = 0,2, \dots, n$ are called weights, which depend only on abscissa, $x_0 = a, x_n = b, h = (x_i - x_{i-1}) = \frac{(b-a)}{n}$. Approximating $f(x)$ by Newton's forward interpolation polynomial of order n or Lagrange interpolation polynomial of order n, we get different weights for different n, giving different Quadrature Rules. Quadrature rules and corresponding weights for some values of n are listed in the below table:

n	Quadrature rule	Weights	Formula
$n = 1$	**Trapezoidal Rule**	$\frac{h}{2}[1, 1]$	$\int_{x_0}^{x_1} y dx = \frac{h}{2}\{y_0 + y_1\}$
$n = 2$	**Simpson's 1/3rd Rule**	$\frac{h}{3}[1, 4, 1]$	$\int_{x_0}^{x_2} y dx = \frac{h}{3}\{y_0 + 4y_1 + y_2\}$
$n = 3$	**Simpson's 3/8th Rule**	$\frac{3h}{8}[1, 3, 3, 1]$	$\int_{x_0}^{x_3} y dx = \frac{3h}{8}\{y_0 + 3y_1 + 3y_2 + y_3\}$
$n = 6$	**Weddle's Rule**	$\frac{3h}{10}[1, 5, 1, 6, 1, 5, 1]$	$\int_{x_0}^{x_3} y dx = \frac{3h}{10}\{y_0 + 5y_1 + y_2 + 6y_3 + y_4 + 5y_5 + y_6\}$

Composite Quadrature Rules for above Rules are obtained by adding quadrature formulas applied for two or more consecutive intervals of equal lengths. Composite Trapezoidal rule and Composite Weddle's rules are given below:

Composite Trapezoidal Rule (Number of subintervals can be any positive integer)		
Number of subintervals	**Weights**	**Formula**
2 (2+1=3 points)	$\frac{h}{2}[1,2,1]$	$\int_{x_0}^{x_2} ydx = \frac{h}{2}\{y_0 + 2y_1 + y_2\}$
3 (3+1=4 points)	$\frac{h}{2}[1,2,2,1]$	$\int_{x_0}^{x_3} ydx = \frac{h}{2}\{y_0 + 2y_1 + 2y_2 + y_3\}$
4 (4+1=5 points)	$\frac{h}{2}[1,2,2,2,1]$	$\int_{x_0}^{x_4} ydx = \frac{h}{2}\{y_0 + 2y_1 + 2y_2 + 2y_3 + y_4\}$
5 (5+1=6 points)	$\frac{h}{2}[1,2,2,2,2,1]$	$\int_{x_0}^{x_5} ydx = \frac{h}{2}\{y_0 + 2y_1 + 2y_2 + 2y_3 + 2y_4 + y_5\}$
6 (6+1=7 points)	$\frac{h}{2}[1,2,2,2,2,2,1]$	$\int_{x_0}^{x_6} ydx = \frac{h}{2}\{y_0 + 2y_1 + 2y_2 + 2y_3 + 2y_4 + 2y_5 + y_6\}$
Composite Weddle's Rule (Number of sub-intervals should be multiple of 6)		
6 (6+1=7 points)	$\frac{3h}{10}[1,5,1,6,1,5,1]$	$\int_{x_0}^{x_6} ydx = \frac{3h}{10}\{y_0 + 5y_1 + y_2 + 6y_3 + y_4 + 5y_5 + y_6\}$
12 (12+1=13 points)	$\frac{3h}{10}[1,5,1,6,1,5,2,5,1,6,1,5,1]$	$\int_{x_0}^{x_{12}} ydx = \frac{3h}{10}\{y_0 + 5y_1 + y_2 + 6y_3 + y_4 + 5y_5 + 2y_6 + 5y_7 + y_8 + 6y_9 + y_{10} + 5y_{11} + y_{12}\}$

Program: (Trapezoidal Rule)

Program to evaluate $\int_a^b f(x)dx$ from given data points by Trapezoidal rule.

Note that $f(x)$ may be unknown or known but data is given.

x	x_0	x_1	x_2	x_3	x_4	x_5	x_6
y	y_0	y_1	y_2	y_3	y_4	y_5	y_6

Program:

```
kill(all)$
fpprintprec:5$
X:[given values of x separated by comma]$
Y:[given values of y separated by comma]$
a:first(X)$
b:last(X)$
h:abs(first(X)-second(X))$
I:h/2*[1,2,2,2,2,2,1].Y$
print("Given Data Table is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("By Trapezoidal rule")$
'integrate(f(x),x,a,b)=I;
```

Program to evaluate $\int_a^b f(x)dx$ from given $f(x)$ by Trapezoidal rule.

Program:

```
kill(all)$
fpprintprec:6$
f(x):=given function$
a:lower limit of integral$
b:upper limit of integral$
n:6$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/2*[1,2,2,2,2,2,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Trapezoidal rule")$
'integrate(f(x),x,a,b)=I;
```

Note: Use weight vector of appropriate length. In the above illustration weight vector of length 7 is used for 7 data points (i.e. for n=6).

Program: (Weddle's Rule)

Program to evaluate $\int_a^b f(x)dx$ from given data points by Weddle's rule.

Note that $f(x)$ may be unknown or known but data is given.

x	x_0	x_1	x_2	x_3	x_4	x_5	x_6
y	y_0	y_1	y_2	y_3	y_4	y_5	y_6

Program:

```
kill(all)$
fpprintprec:5$
X:[given values of x separated by comma]$
Y:[given values of y separated by comma]$
a:first(X)$
b:last(X)$
h:abs(first(X)-second(X))$
I:3*h/10*[1,5,1,6,1,5,1].Y$
print("Given Data Table is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("By Weddle's rule")$
'integrate(f(x),x,a,b)=I;
```

Program to evaluate $\int_a^b f(x)dx$ from given $f(x)$ by Weddle's rule.

Program:

```
kill(all)$
fpprintprec:6$
f(x):=given function$
a:lower limit of integral$
b:upper limit of integral$
n:6$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:3*h/10*[1,5,1,6,1,5,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Weddle's rule")$
'integrate(f(x),x,a,b)=I;
```

Note: Use weight vector of appropriate length. In the above illustration weight vector of length 7 is used for 7 data points (i.e. for n=6).

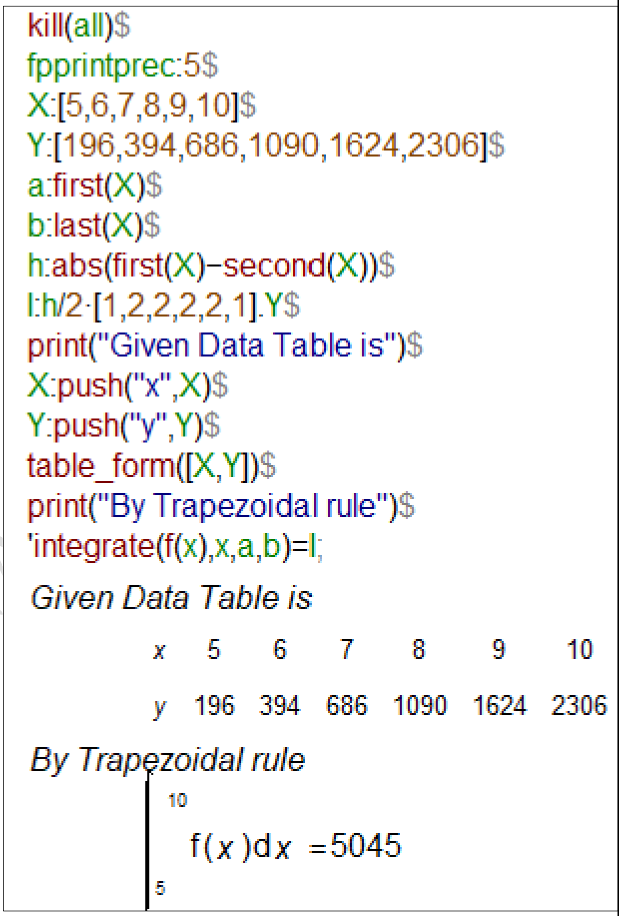
Worked Examples: (Trapezoidal Rule)

Problem 1. Write a program to evaluate the integral $\int_5^{10} f(x)dx$ from the given data points by Trapezoidal rule.

x	5	6	7	8	9	10
y	196	394	686	1090	1624	2306

Program:

```
kill(all)$
fpprintprec:5$
X:[5,6,7,8,9,10]$
Y:[196,394,686,1090,1624,2306]$
a:first(X)$
b:last(X)$
h:abs(first(X)-second(X))$
I:h/2*[1,2,2,2,2,1].Y$
print("Data Table for integration is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("By Trapezoidal rule")$
'integrate(f(x),x,a,b)=I;
```



```
kill(all)$
fpprintprec:5$
X:[5,6,7,8,9,10]$
Y:[196,394,686,1090,1624,2306]$
a:first(X)$
b:last(X)$
h:abs(first(X)−second(X))$
I:h/2·[1,2,2,2,2,1].Y$
print("Given Data Table is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("By Trapezoidal rule")$
'integrate(f(x),x,a,b)=I;
```

Given Data Table is

x	5	6	7	8	9	10
y	196	394	686	1090	1624	2306

By Trapezoidal rule

$$\int_5^{10} f(x)dx = 5045$$

Output:

Given Data Table is

x	5	6	7	8	9	10
y	196	394	686	1090	1624	2306

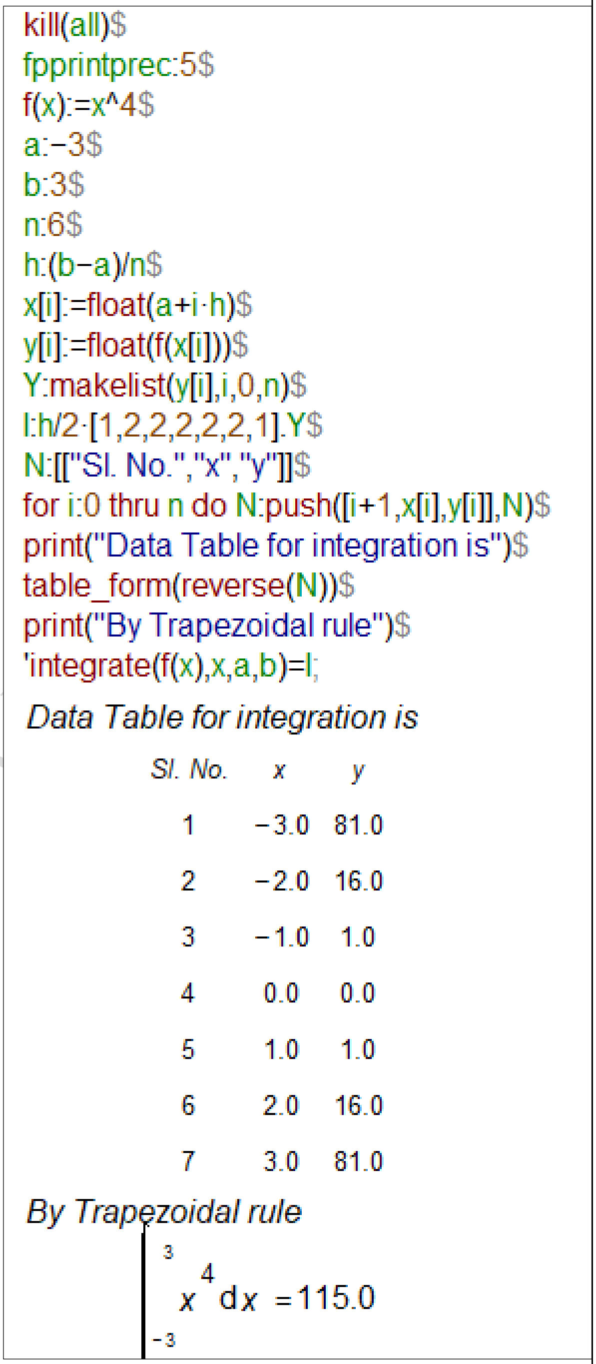
By Trapezoidal rule

$$\int_5^{10} f(x)\,dx = 5045$$

Problem 2. Write a program to evaluate the integral $\int_{-3}^{3} x^4 dx$ by Trapezoidal rule by taking 6 subintervals.

Program:

```
kill(all)$
fpprintprec:5$
f(x):=x^4$
a:-3$
b:3$
n:6$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/2*[1,2,2,2,2,2,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Trapezoidal rule")$
'integrate(f(x),x,a,b)=I;
```



Output:

Data Table for integration is

Sl. No.	*x*	*y*
1	−3.0	81.0
2	−2.0	16.0
3	−1.0	1.0
4	0.0	0.0
5	1.0	1.0
6	2.0	16.0
7	3.0	81.0

By Trapezoidal rule

$$\int_{-3}^{3} x^4 dx = 115.0$$

Problem 3. Write a program to evaluate the integral $\int_0^1 e^{-x^2}\,dx$ by Trapezoidal rule by taking 10 subintervals.

Program:

```
kill(all)$
fpprintprec:5$
f(x):=exp(-x^2)$
a:0$
b:1$
n:10$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/2*[1,2,2,2,2,2,2,2,2,2,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Trapezoidal rule")$
'integrate(f(x),x,a,b)=I;
```

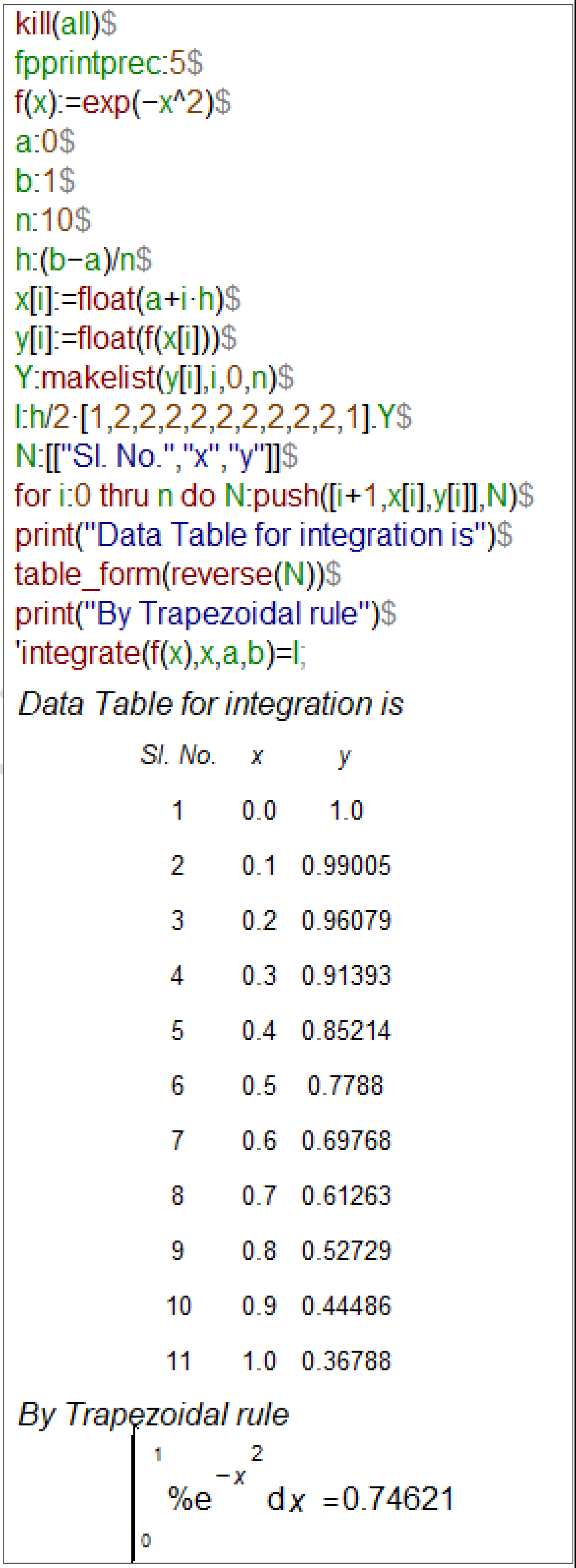
Output:

Data Table for integration is

$Sl. No.$	x	y
1	0.0	1.0
2	0.1	0.99005
3	0.2	0.96079
4	0.3	0.91393
5	0.4	0.85214
6	0.5	0.7788
7	0.6	0.69768
8	0.7	0.61263
9	0.8	0.52729
10	0.9	0.44486
11	1.0	0.36788

By Trapezoidal rule

$$\int_0^1 \%e^{-x^2}\,dx = 0.74621$$



```
kill(all)$
fpprintprec:5$
f(x):=exp(-x^2)$
a:0$
b:1$
n:10$
h:(b-a)/n$
x[i]:=float(a+i·h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/2·[1,2,2,2,2,2,2,2,2,2,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Trapezoidal rule")$
'integrate(f(x),x,a,b)=I;
```

Data Table for integration is

Sl. No.	x	y
1	0.0	1.0
2	0.1	0.99005
3	0.2	0.96079
4	0.3	0.91393
5	0.4	0.85214
6	0.5	0.7788
7	0.6	0.69768
8	0.7	0.61263
9	0.8	0.52729
10	0.9	0.44486
11	1.0	0.36788

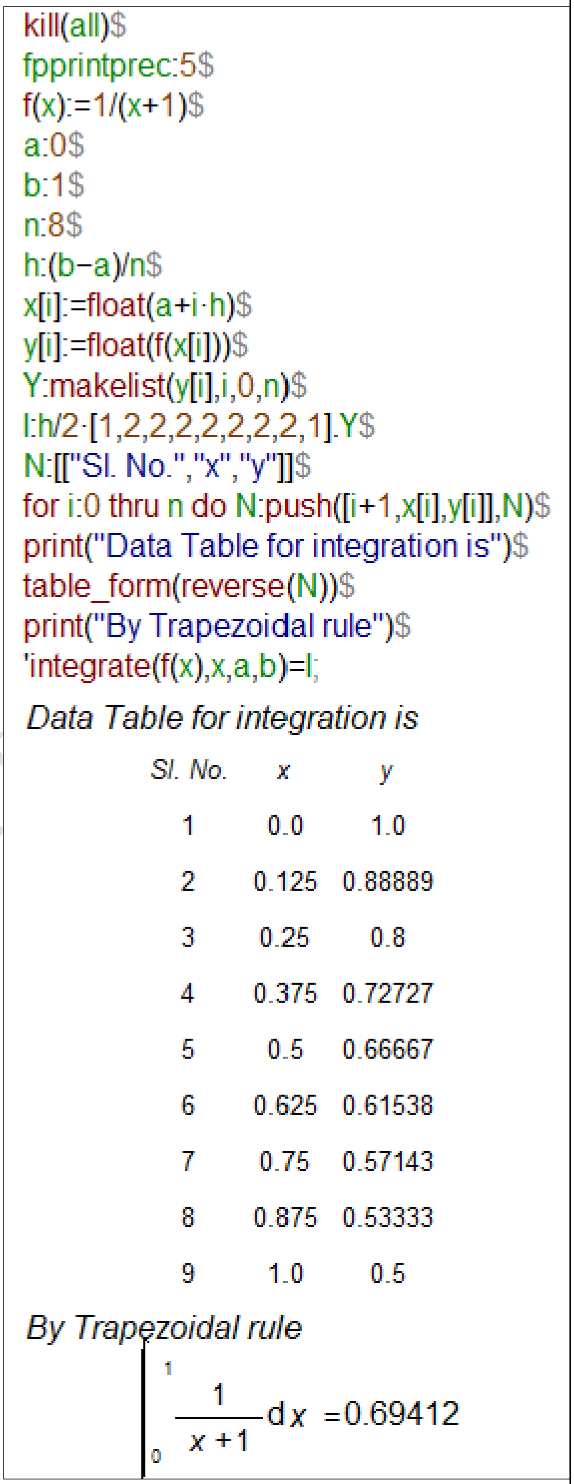
By Trapezoidal rule

$$\int_0^1 \%e^{-x^2}\,dx = 0.74621$$

Problem 4. Write a program to evaluate the integral $\int_0^1 \frac{1}{x+1}dx$ by Trapezoidal rule by taking 9 ordinates.

Program:

```
kill(all)$
fpprintprec:5$
f(x):=1/(x+1)$
a:0$
b:1$
n:8$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:h/2*[1,2,2,2,2,2,2,2,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Trapezoidal rule")$
'integrate(f(x),x,a,b)=I;
```



Output:

Data Table for integration is

$Sl.No.$	x	y
1	0.0	1.0
2	0.125	0.88889
3	0.25	0.8
4	0.375	0.72727
5	0.5	0.66667
6	0.625	0.61538
7	0.75	0.57143
8	0.875	0.53333
9	1.0	0.5

By Trapezoidal rule

$$\int_0^1 \frac{1}{x+1}dx = 0.69412$$

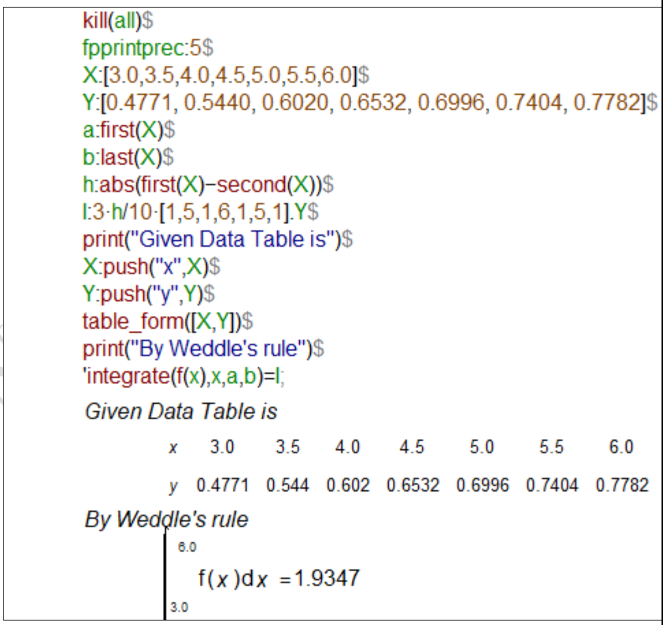
Worked Examples: (Weddle's Rule)

Problem 5. Write a program to evaluate the integral $\int_{3.0}^{6.0} f(x)\,dx$ from the given data points by Weddle's rule.

x	3.0	3.5	4.0	4.5	5.0	5.5	6.0
y	0.4771	0.5440	0.6020	0.6532	0.6996	0.7404	0.7782

Program:

```
kill(all)$
fpprintprec:5$
X:[3.0,3.5,4.0,4.5,5.0,5.5,6.0]$
Y:[0.4771, 0.5440, 0.6020, 0.6532, 0.6996, 0.7404, 0.7782]$
a:first(X)$
b:last(X)$
h:abs(first(X)-second(X))$
I:3*h/10*[1,5,1,6,1,5,1].Y$
print("Given Data Table is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("By Weddle's rule")$
'integrate(f(x),x,a,b)=I;
```



```
kill(all)$
fpprintprec:5$
X:[3.0,3.5,4.0,4.5,5.0,5.5,6.0]$
Y:[0.4771, 0.5440, 0.6020, 0.6532, 0.6996, 0.7404, 0.7782]$
a:first(X)$
b:last(X)$
h:abs(first(X)−second(X))$
I:3·h/10·[1,5,1,6,1,5,1].Y$
print("Given Data Table is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("By Weddle's rule")$
'integrate(f(x),x,a,b)=I;
```

Given Data Table is

x	3.0	3.5	4.0	4.5	5.0	5.5	6.0
y	0.4771	0.544	0.602	0.6532	0.6996	0.7404	0.7782

By Weddle's rule

$$\int_{3.0}^{6.0} f(x)\,dx = 1.9347$$

Output:

$Given\ Data\ Table\ is$

x	3.0	3.5	4.0	4.5	5.0	5.5	6.0
y	0.4771	0.544	0.602	0.6532	0.6996	0.7404	0.7782

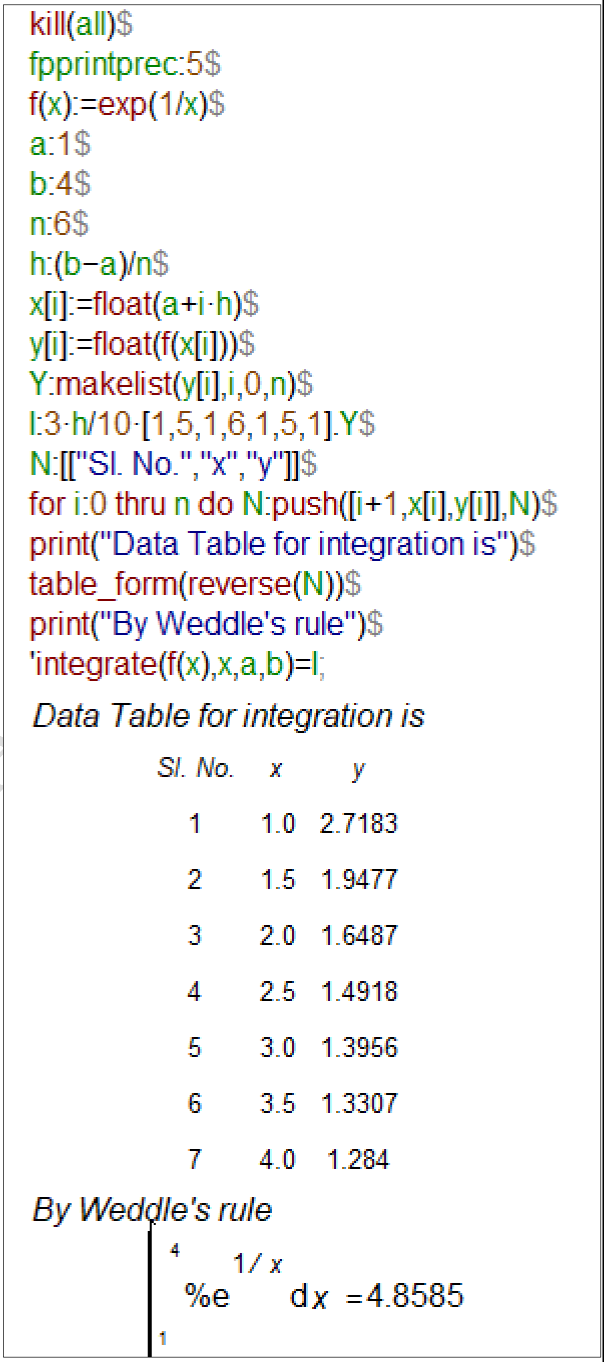
$By\ Weddle's\ rule$

$$\int_{3.0}^{6.0} f(x)\,dx = 1.9347$$

Problem 6. Write a program to evaluate the integral $\int_1^4 e^{\frac{1}{x}}\,dx$ by Weddle’s rule by taking 6 subintervals.

Program:

```
kill(all)$
fpprintprec:5$
f(x):=exp(1/x)$
a:1$
b:4$
n:6$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:3*h/10*[1,5,1,6,1,5,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Weddle's rule")$
'integrate(f(x),x,a,b)=I;
```



```
kill(all)$
fpprintprec:5$
f(x):=exp(1/x)$
a:1$
b:4$
n:6$
h:(b−a)/n$
x[i]:=float(a+i·h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:3·h/10·[1,5,1,6,1,5,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Weddle's rule")$
'integrate(f(x),x,a,b)=I;
```

Data Table for integration is

Sl. No.	x	y
1	1.0	2.7183
2	1.5	1.9477
3	2.0	1.6487
4	2.5	1.4918
5	3.0	1.3956
6	3.5	1.3307
7	4.0	1.284

By Weddle's rule

$$\int_1^4 \%e^{1/x}\,dx = 4.8585$$

Output:

Data Table for integration is

Sl. No.	x	y
1	1.0	2.7183
2	1.5	1.9477
3	2.0	1.6487
4	2.5	1.4918
5	3.0	1.3956
6	3.5	1.3307
7	4.0	1.284

By Weddle's rule

$$\int_1^4 \%e^{\frac{1}{x}}\,dx = 4.8585$$

Problem 7. Write a program to evaluate the integral $\int_0^6 \frac{dx}{1+x^2}$ by Weddle's rule by taking 12 subintervals.

Program:

```
kill(all)$
fpprintprec:5$
f(x):=1/(1+x^2)$
a:0$
b:6$
n:12$
h:(b-a)/n$
x[i]:=float(a+i*h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:3*h/10*[1,5,1,6,1,5,2,5,1,6,1,5,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Weddle's rule")$
'integrate(f(x),x,a,b)=I;
```

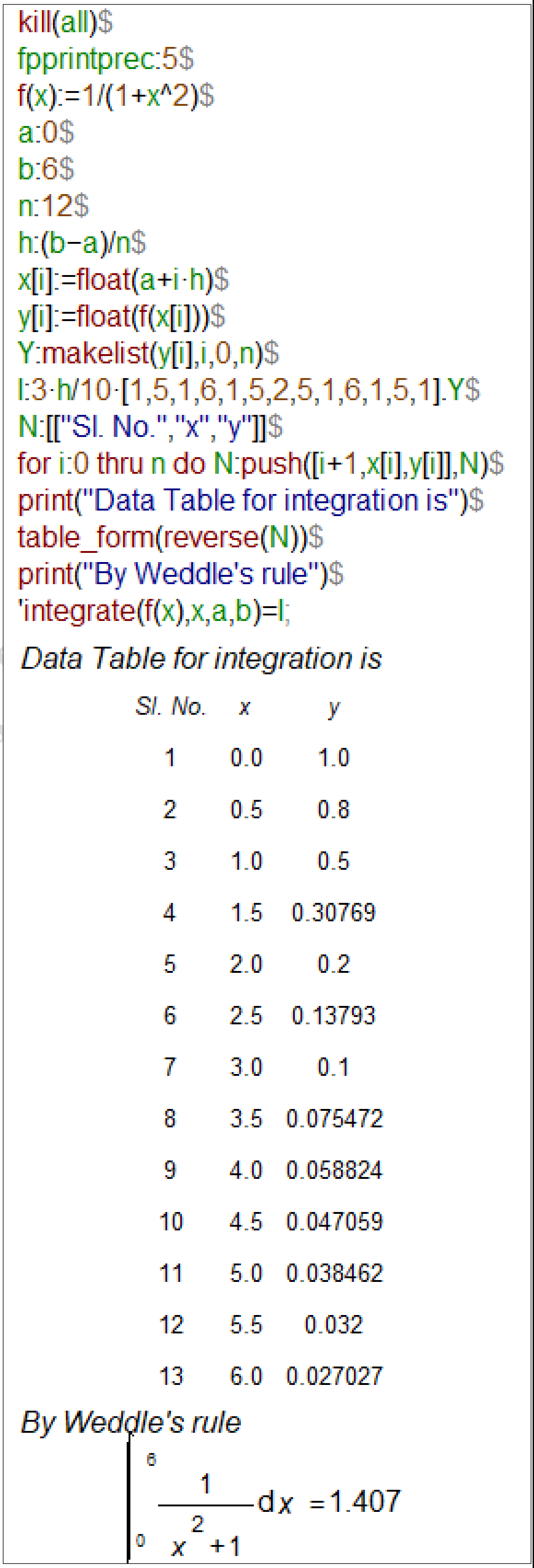
Output:

Data Table for integration is

$Sl.No.$	x	y
1	0.0	1.0
2	0.5	0.8
3	1.0	0.5
4	1.5	0.30769
5	2.0	0.2
6	2.5	0.13793
7	3.0	0.1
8	3.5	0.075472
9	4.0	0.058824
10	4.5	0.047059
11	5.0	0.038462
12	5.5	0.032
13	6.0	0.027027

By Weddle's rule

$$\int_0^6 \frac{1}{x^2+1}\,dx = 1.407$$



```
kill(all)$
fpprintprec:5$
f(x):=1/(1+x^2)$
a:0$
b:6$
n:12$
h:(b−a)/n$
x[i]:=float(a+i·h)$
y[i]:=float(f(x[i]))$
Y:makelist(y[i],i,0,n)$
I:3·h/10·[1,5,1,6,1,5,2,5,1,6,1,5,1].Y$
N:[["Sl. No.","x","y"]]$
for i:0 thru n do N:push([i+1,x[i],y[i]],N)$
print("Data Table for integration is")$
table_form(reverse(N))$
print("By Weddle's rule")$
'integrate(f(x),x,a,b)=I;
```

Data Table for integration is

Sl. No.	x	y
1	0.0	1.0
2	0.5	0.8
3	1.0	0.5
4	1.5	0.30769
5	2.0	0.2
6	2.5	0.13793
7	3.0	0.1
8	3.5	0.075472
9	4.0	0.058824
10	4.5	0.047059
11	5.0	0.038462
12	5.5	0.032
13	6.0	0.027027

By Weddle's rule

$$\int_0^6 \frac{1}{x^2+1}\,dx = 1.407$$

Exercise:

I. Write a program to evaluate given definite integrals by Trapezoidal rule.

1. Evaluate $\int_{4.0}^{5.2} f(x)dx$ from given data points

x	4.0	4.2	4.4	4.6	4.8	5.0	5.2
y	1.386	1.435	1.482	1.526	1.569	1.609	1.649

(Answer: 1.8277)

2. Evaluate $\int_0^1 \frac{1}{(1+x)^2} dx$ taking 10 sub-intervals

(Answer: 0.78498)

3. Evaluate $\int_0^1 e^x\, dx$ taking 5 sub-intervals

(Answer: 1.724)

4. Evaluate $\int_0^{\frac{\pi}{2}} e^{\sin(x)} dx$ taking 4 sub-intervals

(Answer: 3.0915)

II. Write a program to evaluate given definite integrals by Weddle's rule.

1. Evaluate $\int_{0.2}^{1.4} f(x)dx$ from given data points

x	0.2	0.4	0.6	0.8	1.0	1.2	1.4
y	0.199	0.389	0.565	0.717	0.841	0.932	0.985

(Answer: 0.80982)

2. Evaluate $\displaystyle\int_0^1 \frac{x}{x^2+1} dx$ taking 6 sub-intervals

(Answer: 0.34657)

3. Evaluate $\int_0^1 e^{-x^2}\, dx$ taking 12 sub-intervals

(Answer: 0.74682)

4. Evaluate $\int_0^{\frac{\pi}{2}} \sqrt{\sin(x)}\, dx$ taking 12 sub-intervals

(Answer: 1.195)

Program 10

Program to find differentiation at specified point using Newton-Gregory interpolation method.

Aim: To find first and second derivatives from given data points of an unknown function at specified point using Newton-Gregory interpolation method using Mathematics Softwares (FOSS).

Software: Maxima

Keys:

Key	Function
kill (all)	Unbinds all items on all infolists
product (expr, i, i_0, i_1)	Represents a product of the values of expr as the index i varies from i_0 to i_1.
. (dot)	The operator . represents noncommutative multiplication and scalar product
fpprintprec	This is an option variable to decide the number of digits to print when printing an ordinary float or bigfloat number. Default value is 16. Set any integer from 2 to 16.
:=	The function definition operator
'	The single quote operator ' prevents evaluation.
abs(x)	Absolute value of the number x
L[i]	Subscript operator for L_i
[a_1, a_2,…,a_m]	List of numbers/objects a_1, a_2,…,a_m.
length(list)	Length/ the total number of elements in list.
if cond_1 *then* expr_1 *else* expr_0	evaluates to expr_1 *if* cond_1 evaluates to true, otherwise the expression evaluates to expr_0.
print (“*text*”, *expr*)$	Displays *text* within inverted commas and evaluates and displays *expr*
block ([v_1, …, v_m], expr_1, …, expr_n)	The function *block* allows to make the variables *v_1*, …, *v_m* to be local for a sequence of commands.
push *(item, list)*	*push* prepends the item *item* to the list *list* and returns a copy of the new list
reverse *(list)*	Reverses the order of the members of the *list* (not the members themselves)
table_form()	Displays a 2D list in a form that is more readable than the output from *Maxima*’s default output routine. The input is a list of one or more lists.
diff (expr, x, n)	Returns the n-th derivative of expr with respect to the variable x
first (list)	Returns the first element of a list
last(list)	Returns the last element of a list
second (list)	Returns the second element of a list
makelist *(expr, i, i_0, i_max)*	Returns the list of elements obtained when ev (*expr*, *i=j*) is applied to the elements *j* of the sequence: *i_0*, *i_0 + 1*, *i_0* + 2, ..., with |*j*| ≤ |*i_max*|.
at (expr, eqn)	Evaluates the expression expr with the variables assuming the values as specified for them in equation eqn.

Note:1. Press Shift+Enter for evaluation of commands and display of output.

2. Replace semicolon (;) by dollar ($) to suppress output of any input line and vice-versa.

3. Start each session with kill(all)$ or quit()$ to remove previously assigned values of all symbols

Definitions and Formulae:

Numerical Differentiation: The process of calculating the value of the derivative of a function at some assigned value of x from the given set of values (x_i, y_i). To compute $\frac{dy}{dx}$, unknown function $y = f(x)$ is approximated by suitable interpolating polynomial and then it is differentiated as many times as required. The choice of interpolating polynomial depends on spacing of abscissa x_i (equally spaced or not) and on the assigned value of x at which $\frac{dy}{dx}$ is required.

Derivatives using Newton-Gregory Forward Difference Formula: The Newton-Gregory Forward Difference (NGFD) Interpolation formula is applied when the values of x are equally spaced (i.e., $x_i = x_0 + ih$), $i = 0,1,2, \dots, n$ and $\frac{dy}{dx}$ is required near the beginning of the table. NGFD interpolation formula is given by:

$$\boldsymbol{y} = y_0 + p\Delta y_0 + \frac{p(p-1)}{2!}\Delta^2 y_0 + \frac{p(p-1)(p-2)}{3!}\Delta^3 y_0 + \frac{p(p-1)(p-2)(p-3)}{4!}\Delta^4 y_0 + \cdots$$

i.e.,

$$\boldsymbol{y} = \left[1, p, \frac{p(p-1)}{2!}, \frac{p(p-1)(p-2)}{3!}, \frac{p(p-1)(p-2)(p-3)}{4!}, \dots\right] \cdot [y_0, \Delta y_0, \Delta^2 y_0, \Delta^3 y_0, \Delta^4 y_0, \dots]$$

Where $p = \frac{(x-x_0)}{h}$. Differentiating above formula w.r.t. p, we get,

$$\frac{dy}{dx} = \frac{\mathbf{1}}{\boldsymbol{h}}\left[\Delta y_0 + \frac{(2p-1)}{2!}\Delta^2 y_0 + \frac{(3p^2-6p+2)}{3!}\Delta^3 y_0 + \frac{(4p^3-18p^2+22p-6)}{4!}\Delta^4 y_0 + \cdots\right]$$

i.e.,

$$\frac{dy}{dx} = \frac{\mathbf{1}}{\boldsymbol{h}}\left[0, 1, \frac{(2p-1)}{2!}, \frac{(3p^2-6p+2)}{3!}, \frac{(4p^3-18p^2+22p-6)}{4!}, \dots\right] \cdot [y_0, \Delta y_0, \Delta^2 y_0, \Delta^3 y_0, \Delta^4 y_0, \dots]$$

and

$$\frac{d^2y}{dx^2} = \frac{\mathbf{1}}{\boldsymbol{h}^2}\left[\Delta^2 y_0 + (p-1)\Delta^3 y_0 + \frac{(12p^2-36p+22)}{4!}\Delta^4 y_0 + \cdots\right]$$

i.e.,

$$\frac{d^2y}{dx^2} = \frac{\mathbf{1}}{\boldsymbol{h}^2}\left[0, 0, 1, (p-1), \frac{(12p^2-36p+22)}{4!}, \dots\right] \cdot [y_0, \Delta y_0, \Delta^2 y_0, \Delta^3 y_0, \Delta^4 y_0, \dots]$$

and so on. For table value, x_0, take $p = 0$. If x is near to x_0, take $p = \frac{(x-x_0)}{h}$.

Note: Taking

$$w = \left[1, p, \frac{p(p-1)}{2!}, \frac{p(p-1)(p-2)}{3!}, \frac{p(p-1)(p-2)(p-3)}{4!}, \ldots\right]$$

and

$$M = [y_0, \Delta y_0, \Delta^2 y_0, \Delta^3 y_0, \Delta^4 y_0, \ldots]$$

We get

$$y = w \cdot M$$

and therefore,

$$\frac{dy}{dx} = \frac{\frac{dw}{dp} \cdot M}{h}$$

$$\frac{d^2y}{dx^2} = \frac{\frac{d^2w}{dp^2} \cdot M}{h^2}$$

Derivatives using Newton-Gregory Backward Difference Formula: The Newton-Gregory Backward Difference (NGBD) Interpolation formula is applied when the values of x are equally spaced (i.e., $x_i = x_0 + ih$), $i = 0,1,2, \ldots, n$ and $\frac{dy}{dx}$ is required near the end of the table. NGBD interpolation formula is given by:

$$\boldsymbol{y} = y_n + p\nabla y_n + \frac{p(p+1)}{2!}\nabla^2 y_n + \frac{p(p+1)(p+2)}{3!}\nabla^3 y_n + \frac{p(p+1)(p+2)(p+3)}{4!}\nabla^4 y_n + \cdots$$

i.e.,

$$\boldsymbol{y} = \left[1, p, \frac{p(p+1)}{2!}, \frac{p(p+1)(p+2)}{3!}, \frac{p(p+1)(p+2)(p+3)}{4!}, \ldots\right] \cdot [y_0, \nabla y_0, \nabla^2 y_0, \nabla^3 y_0, \nabla^4 y_0, \ldots]$$

Where $p = \frac{(x-x_n)}{h}$. Differentiating above formula w.r.t. p, we get,

$$\frac{dy}{dx} = \frac{\mathbf{1}}{\boldsymbol{h}}\left[\nabla y_0 + \frac{(2p+1)}{2!}\nabla^2 y_0 + \frac{(3p^2+6p+2)}{3!}\nabla^3 y_0 + \frac{(4p^3+18p^2+22p+6)}{4!}\nabla^4 y_0 + \cdots\right]$$

i.e.,

$$\frac{dy}{dx} = \frac{\mathbf{1}}{\boldsymbol{h}}\left[0, 1, \frac{(2p+1)}{2!}, \frac{(3p^2+6p+2)}{3!}, \frac{(4p^3+18p^2+22p+6)}{4!}, \ldots\right] \cdot [y_0, \nabla y_0, \nabla^2 y_0, \nabla^3 y_0, \nabla^4 y_0, \ldots]$$

and

$$\frac{d^2y}{dx^2} = \frac{\mathbf{1}}{\boldsymbol{h^2}}\left[\nabla^2 y_0 + (p+1)\Delta^3 y_0 + \frac{(12p^2+36p+22)}{4!}\nabla^4 y_0 + \cdots\right]$$

i.e.,

$$\frac{d^2y}{dx^2} = \frac{\mathbf{1}}{\boldsymbol{h^2}}\left[0,0,1,(p+1),\frac{(12p^2+36p+22)}{4!},\dots\right]\cdot[y_0,\nabla y_0,\nabla^2 y_0,\nabla^3 y_0,\nabla^4 y_0,\dots]$$

and so on. For table value, x_n, take $p = 0$. If x is near to x_n, take $p = \frac{(x-x_n)}{h}$.

Note: Taking

$$w = \left[1,p,\frac{p(p+1)}{2!},\frac{p(p+1)(p+2)}{3!},\frac{p(p+1)(p+2)(p+3)}{4!},\dots\right]$$

and

$$M = [y_0,\nabla y_0,\nabla^2 y_0,\nabla^3 y_0,\nabla^4 y_0,\dots]$$

We get

$$y = w\cdot M$$

and therefore,

$$\frac{dy}{dx} = \frac{\frac{dw}{dp}\cdot M}{h}$$

$$\frac{d^2y}{dx^2} = \frac{\frac{d^2w}{dp^2}\cdot M}{h^2}$$

Program: (Using Newton's Forward Interpolation formula)

1. Program to find $\left(\frac{dy}{dx}\right)_{x=x_0}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_0}$ from the given table of values:

x	x_0	x_1	x_2	x_3	x_4	x_5	. . .	x_n
y	y_0	y_1	y_2	y_3	y_4	y_5	. . .	y_n

Program:

```
kill(all)$
fpprintprec:5$
X:[values of x separated by comma]$
Y:[values of y separated by comma]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p-(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n)$
L:Y$
M:[first(L)]$
for i:1 thru n-1 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(first(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```


2. Program to find $\left(\frac{dy}{dx}\right)_{x=x_0+ph}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_0+ph}$ from the given table of values:

x	x_0	x_1	x_2	x_3	x_4	x_5	. . .	x_n
y	y_0	y_1	y_2	y_3	y_4	y_5	. . .	y_n

Program:

```
kill(all)$
fpprintprec:5$
X:[values of x separated by comma]$
Y:[values of y separated by comma]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p-(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n)$
```

```
L:Y$
M:[first(L)]$
for i:1 thru n-1 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(first(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=(x-x0)/h).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=(x-x0)/h).M/h^2;
```

3. Program to find $\left(\frac{dy}{dx}\right)_{x=x_1}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_1}$ from the given table of values:

x	x_0	x_1	x_2	x_3	x_4	x_5	. . .	x_n
y	y_0	y_1	y_2	y_3	y_4	y_5	. . .	y_n

Program:

```
kill(all)$
fpprintprec:5$
X:[values of x separated by comma]$
Y:[values of y separated by comma]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p-(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n-1)$
L:Y$
M:[second(L)]$
for i:1 thru n-2 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(second(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```

Program: (Using Newton's Backward Interpolation formula)

1. Program to find $\left(\frac{dy}{dx}\right)_{x=x_n}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_n}$ from the given table of values:

x	x_0	x_1	x_2	x_3	x_4	x_5	. . .	x_n
y	y_0	y_1	y_2	y_3	y_4	y_5	. . .	y_n

Program:

```
kill(all)$
fpprintprec:5$
X:[values of x separated by comma]$
Y:[values of y separated by comma]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p+(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n)$
L:Y$
M:[last(L)]$
for i:1 thru n-1 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(last(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```


2. Program to find $\left(\frac{dy}{dx}\right)_{x=x_n+ph}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_n+ph}$ from the given table of values:

x	x_0	x_1	x_2	x_3	x_4	x_5	. . .	x_n
y	y_0	y_1	y_2	y_3	y_4	y_5	. . .	y_n

Program:

```
kill(all)$
fpprintprec:5$
X:[values of x separated by comma]$
Y:[values of y separated by comma]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p+(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n)$
```

```
L:Y$
M:[last(L)]$
for i:1 thru n-1 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(last(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=(x-xn)/h).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=(x-xn)/h).M/h^2;
```

3. Program to find $\left(\frac{dy}{dx}\right)_{x=x_{n-1}}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_{n-1}}$ from the given table of values:

x	x_0	x_1	x_2	x_3	x_4	x_5	. . .	x_n
y	y_0	y_1	y_2	y_3	y_4	y_5	. . .	y_n

Program:

```
kill(all)$
fpprintprec:5$
X:[values of x separated by comma]$
Y:[values of y separated by comma]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p+(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n-1)$
L:Y$
M:[ second(reverse(L)]$
for i:1 thru n-2 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(second(reverse(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```

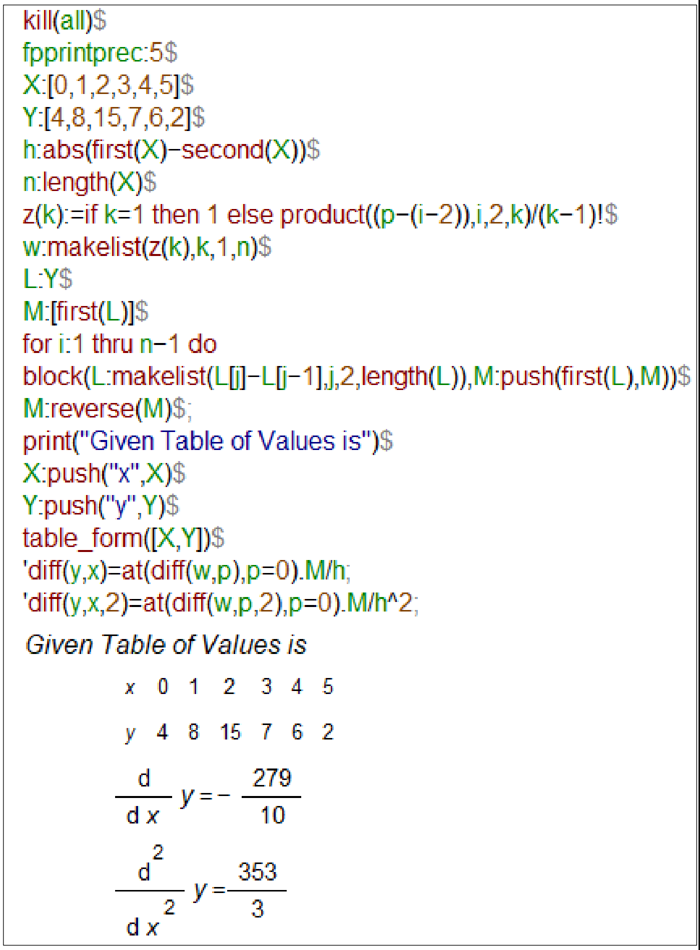
Worked Examples:

Problem 1. Write a program to find $\left(\frac{dy}{dx}\right)_{x=x_0}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_0}$ from the given table of values:

x	0	1	2	3	4	5
y	4	8	15	7	6	2

Program:

```
kill(all)$
fpprintprec:5$
X:[0,1,2,3,4,5]$
Y:[4,8,15,7,6,2]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p-(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n)$
L:Y$
M:[first(L)]$
for i:1 thru n-1 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(first(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```



```
kill(all)$
fpprintprec:5$
X:[0,1,2,3,4,5]$
Y:[4,8,15,7,6,2]$
h:abs(first(X)−second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p−(i−2)),i,2,k)/(k−1)!$
w:makelist(z(k),k,1,n)$
L:Y$
M:[first(L)]$
for i:1 thru n−1 do
block(L:makelist(L[j]−L[j−1],j,2,length(L)),M:push(first(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```

Given Table of Values is

x	0	1	2	3	4	5
y	4	8	15	7	6	2

$$\frac{d}{dx}y=-\frac{279}{10}$$

$$\frac{d^2}{dx^2}y=\frac{353}{3}$$

Output:

Given Table of Values is

x	0	1	2	3	4	5
y	4	8	15	7	6	2

$$\frac{d}{dx}y=-\frac{279}{10}$$

$$\frac{d^2}{dx^2}y=\frac{353}{3}$$

Problem 2. Write a program to find $\left(\frac{dy}{dx}\right)_{x=x_1}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_1}$ from the given table of values:

x	1.0	1.2	1.4	1.6	1.8	2.0	2.2
y	2.7183	3.3201	4.0552	4.9530	6.0496	7.3891	9.0250

Program:

```
kill(all)$
fpprintprec:5$
X:[1.0,1.2,1.4, 1.6, 1.8, 2.0, 2.2]$
Y:[2.7183,3.3201, 4.0552, 4.9530, 6.0496, 7.3891, 9.0250]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p-(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n-1)$
L:Y$
M:[second(L)]$
for i:1 thru n-2 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(second(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```


```
kill(all)$
fpprintprec:5$
X:[1.0,1.2,1.4, 1.6, 1.8, 2.0, 2.2]$
Y:[2.7183,3.3201, 4.0552, 4.9530, 6.0496, 7.3891, 9.0250]$
h:abs(first(X)−second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p−(i−2)),i,2,k)/(k−1)!$
w:makelist(z(k),k,1,n−1)$
L:Y$
M:[second(L)]$
for i:1 thru n−2 do
block(L:makelist(L[j]−L[j−1],j,2,length(L)),M:push(second(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```

Given Table of Values is

x	1.0	1.2	1.4	1.6	1.8	2.0	2.2
y	2.7183	3.3201	4.0552	4.953	6.0496	7.3891	9.025

$\frac{d}{dx} y = 3.3203$

$\frac{d^2}{dx^2} y = 3.3192$

Output:

Given Table of Values is

x	1.0	1.2	1.4	1.6	1.8	2.0	2.2
y	2.7183	3.3201	4.0552	4.953	6.0496	7.3891	9.025

$$\frac{d}{dx} y = 3.3203$$

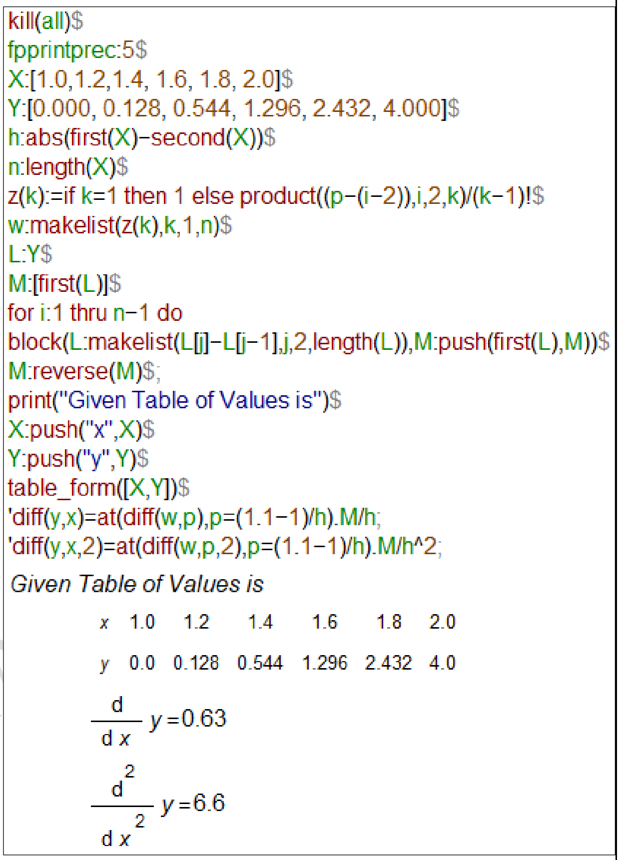
$$\frac{d^2}{dx^2} y = 3.3192$$

Problem 3. Write a program to find $\left(\frac{dy}{dx}\right)_{x=1.1}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=1.1}$ from the given table of values:

x	1.0	1.2	1.4	1.6	1.8	2.0
y	0.000	0.128	0.544	1.296	2.432	4.000

Program:

```
kill(all)$
fpprintprec:5$
X:[1.0,1.2,1.4, 1.6, 1.8, 2.0]$
Y:[0.000, 0.128, 0.544, 1.296, 2.432, 4.000]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p-(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n)$
L:Y$
M:[first(L)]$
for i:1 thru n-1 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(first(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=(1.1-1)/h).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=(1.1-1)/h).M/h^2;
```



```
kill(all)$
fpprintprec:5$
X:[1.0,1.2,1.4, 1.6, 1.8, 2.0]$
Y:[0.000, 0.128, 0.544, 1.296, 2.432, 4.000]$
h:abs(first(X)−second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p−(i−2)),i,2,k)/(k−1)!$
w:makelist(z(k),k,1,n)$
L:Y$
M:[first(L)]$
for i:1 thru n−1 do
block(L:makelist(L[j]−L[j−1],j,2,length(L)),M:push(first(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=(1.1−1)/h).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=(1.1−1)/h).M/h^2;
```

Given Table of Values is

x	1.0	1.2	1.4	1.6	1.8	2.0
y	0.0	0.128	0.544	1.296	2.432	4.0

$$\frac{d}{dx}y=0.63$$

$$\frac{d^2}{dx^2}y=6.6$$

Output:

Given Table of Values is

x	1.0	1.2	1.4	1.6	1.8	2.0
y	0.0	0.128	0.544	1.296	2.432	4.0

$$\frac{d}{dx}y = 0.63$$

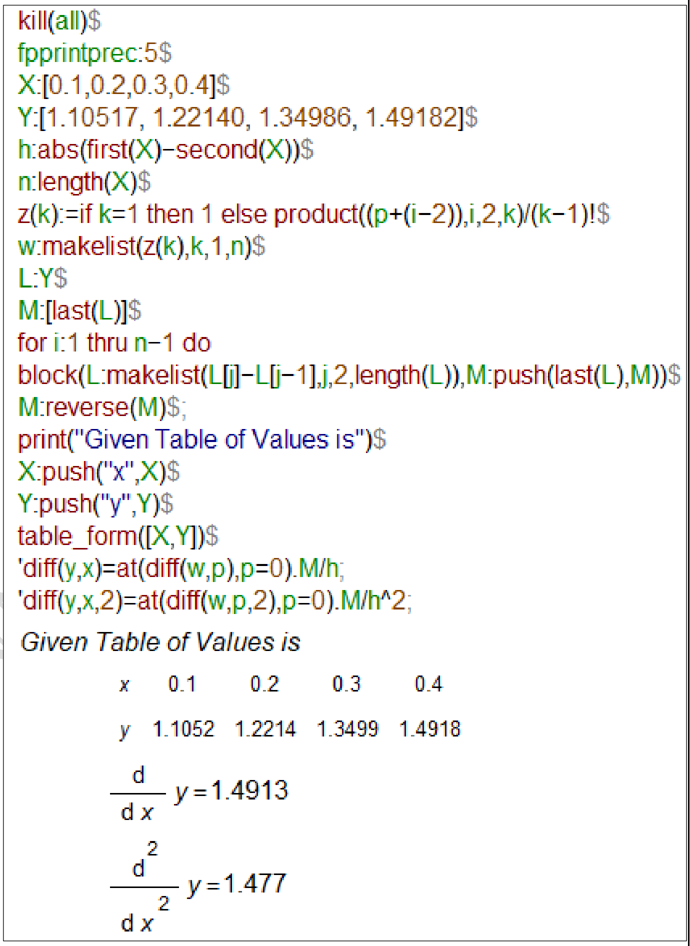
$$\frac{d^2}{dx^2}y = 6.6$$

Problem 4. Write a program to find $\left(\frac{dy}{dx}\right)_{x=x_n}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_n}$ from the given table of values:

x	0.1	0.2	0.3	0.4
y	1.10517	1.22140	1.34986	1.49182

Program:

```
kill(all)$
fpprintprec:5$
X:[0.1,0.2,0.3,0.4]$
Y:[1.10517, 1.22140, 1.34986, 1.49182]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p+(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n)$
L:Y$
M:[last(L)]$
for i:1 thru n-1 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(last(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```



```
kill(all)$
fpprintprec:5$
X:[0.1,0.2,0.3,0.4]$
Y:[1.10517, 1.22140, 1.34986, 1.49182]$
h:abs(first(X)−second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p+(i−2)),i,2,k)/(k−1)!$
w:makelist(z(k),k,1,n)$
L:Y$
M:[last(L)]$
for i:1 thru n−1 do
block(L:makelist(L[j]−L[j−1],j,2,length(L)),M:push(last(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```

Given Table of Values is

x	0.1	0.2	0.3	0.4
y	1.1052	1.2214	1.3499	1.4918

$$\frac{d}{dx}y=1.4913$$

$$\frac{d^2}{dx^2}y=1.477$$

Output:

Given Table of Values is

x	0.1	0.2	0.3	0.4
y	1.1052	1.2214	1.3499	1.4918

$$\frac{d}{dx}y = 1.4913$$

$$\frac{d^2}{dx^2}y = 1.477$$

Problem 5. Write a program to find $\left(\frac{dy}{dx}\right)_{x=x_{n-1}}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_{n-1}}$ from the given table of values:

x	1	2	3	4	5
y	1	7	25	61	121

Program:

```
kill(all)$
fpprintprec:5$
X:[1,2,3,4,5]$
Y:[1,7,25,61,121]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p+(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n-1)$
L:Y$
M:[second(reverse(L))]$
for i:1 thru n-2 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(second(reverse(L)),M))$
M:reverse(M)$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```

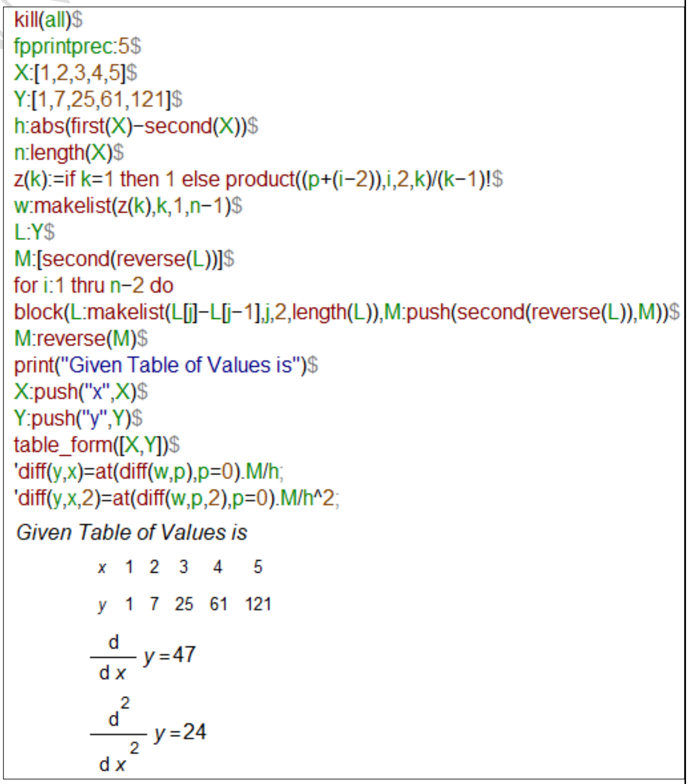
Output:

Given Table of Values is

x	1	2	3	4	5
y	1	7	25	61	121

$$\frac{d}{dx}y = 47$$

$$\frac{d^2}{dx^2}y = 24$$



```
kill(all)$
fpprintprec:5$
X:[1,2,3,4,5]$
Y:[1,7,25,61,121]$
h:abs(first(X)−second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p+(i−2)),i,2,k)/(k−1)!$
w:makelist(z(k),k,1,n−1)$
L:Y$
M:[second(reverse(L))]$
for i:1 thru n−2 do
block(L:makelist(L[j]−L[j−1],j,2,length(L)),M:push(second(reverse(L)),M))$
M:reverse(M)$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=0).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=0).M/h^2;
```

Given Table of Values is

x	1	2	3	4	5
y	1	7	25	61	121

$$\frac{d}{dx}y=47$$

$$\frac{d^2}{dx^2}y=24$$

Problem 6. Write a program to find $\left(\frac{dy}{dx}\right)_{x=2.03}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=2.03}$ from the given table of values:

x	1.96	1.98	2.00	2.02	2.04
y	0.7825	0.7739	0.7651	0.7563	0.7473

Program:

```
kill(all)$
fpprintprec:5$
X:[1.96,1.98,2.00, 2.02, 2.04]$
Y:[0.7825, 0.7739, 0.7651, 0.7563, 0.7473]$
h:abs(first(X)-second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p+(i-2)),i,2,k)/(k-1)!$
w:makelist(z(k),k,1,n)$
L:Y$
M:[last(L)]$
for i:1 thru n-1 do
block(L:makelist(L[j]-L[j-1],j,2,length(L)),M:push(last(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=(2.03-2.04)/h).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=(2.03-2.04)/h).M/h^2;
```

Output:

Given Table of Values is

x	1.96	1.98	2.0	2.02	2.04
y	0.7825	0.7739	0.7651	0.7563	0.7473

$$\frac{d}{dx}y = -0.44875$$

$$\frac{d^2}{dx^2}y = -1.0417$$


```
kill(all)$
fpprintprec:5$
X:[1.96,1.98,2.00, 2.02, 2.04]$
Y:[0.7825, 0.7739, 0.7651, 0.7563, 0.7473]$
h:abs(first(X)−second(X))$
n:length(X)$
z(k):=if k=1 then 1 else product((p+(i−2)),i,2,k)/(k−1)!$
w:makelist(z(k),k,1,n)$
L:Y$
M:[last(L)]$
for i:1 thru n−1 do
block(L:makelist(L[j]−L[j−1],j,2,length(L)),M:push(last(L),M))$
M:reverse(M)$;
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
'diff(y,x)=at(diff(w,p),p=(2.03−2.04)/h).M/h;
'diff(y,x,2)=at(diff(w,p,2),p=(2.03−2.04)/h).M/h^2;
```

Given Table of Values is

x 1.96 1.98 2.0 2.02 2.04
y 0.7825 0.7739 0.7651 0.7563 0.7473

$$\frac{d}{dx}y=-0.44875$$

$$\frac{d^2}{dx^2}y=-1.0417$$

Exercise:

1. Write a program to find $\left(\frac{dy}{dx}\right)_{x=x_n}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_n}$ from the given table of values:

x	0	21	2	3	4	5
y	4	8	15	7	6	2

(Answer: $\frac{d}{dx}y = -\frac{937}{30}, \frac{d^2}{dx^2}y = -\frac{307}{3}$)

2. Write a program to find $\left(\frac{dy}{dx}\right)_{x=x_0}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_0}$ from the given table of values:

x	5	6	9	11
y	12	13	14	16

(Answer: $\frac{d}{dx}y = -\frac{279}{10}, \frac{d^2}{dx^2}y = \frac{353}{3}$)

3. Write a program to find $\left(\frac{dy}{dx}\right)_{x=x_0}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=x_0}$ from the given table of values:

x	0.1	0.2	0.3	0.4
y	1.10517	1.22140	1.34986	1.49182

(Answer: $\frac{d}{dx}y = 1.1054, \frac{d^2}{dx^2}y = 1.096$)

4. Write a program to find $\left(\frac{dy}{dx}\right)_{x=1.1}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=1.1}$ from the given table of values:

x	1.0	1.1	1.2	1.3	1.4	1.5	1.6
y	7.989	8.403	8.781	9.129	9.451	9.750	10.031

(Answer: $\frac{d}{dx}y = 3.9518, \frac{d^2}{dx^2}y = -3.7417$)

5. Write a program to find $\left(\frac{dy}{dx}\right)_{x=8}$ and $\left(\frac{d^2y}{dx^2}\right)_{x=8}$ from the given table of values:

x	1	3	5	7	9
y	85.3	74.5	67.0	60.5	54.3

(Answer: $\frac{d}{dx}y = -3.11875, \frac{d^2}{dx^2}y = 0.10416667$)

Program 11

Program to find the missing value of table using Lagrange method.

Aim: To find interpolating polynomial and the missing value of table using Lagrange method using Mathematics Softwares (FOSS).

Software: Maxima

Keys:

Key	Function
kill (all)	Unbinds all items on all infolists
load ("interpol")	Loads package interpol which defines the Lagrangian, the linear and the cubic splines methods for polynomial interpolation.
lagrange *(points)*	Computes the polynomial interpolation by the Lagrangian method. Argument points must be either: • a two column matrix, p:matrix([2,4],[5,6],[9,3]) • a list of pairs, p: [[2,4],[5,6],[9,3]] Default variable is *x*. Note that when working with high degree polynomials, floating point evaluations are unstable.
makelist *(expr, i, i_0, i_max)*	Returns the list of elements obtained when ev (*expr*, *i=j*) is applied to the elements *j* of the sequence: *i_0*, *i_0* + *1*, *i_0* + 2, ..., with |*j*| ≤ |*i_max*|.
at (expr, eqn)	Evaluates the expression expr with the variables assuming the values as specified for them in equation eqn.
:=	The function definition operator
[$a_1, a_2, \ldots, a_m$]	List of numbers/objects $a_1, a_2, \ldots, a_m$.
if cond_1 *then* expr_1 *else* expr_0	evaluates to expr_1 *if* cond_1 evaluates to true, otherwise the expression evaluates to expr_0.
print (“*text*”, *expr*)$	Displays *text* within inverted commas and evaluates and displays *expr*
push *(item, list)*	*push* prepends the item *item* to the list *list* and returns a copy of the new list
table_form()	Displays a 2D list in a form that is more readable than the output from *Maxima*’s default output routine. The input is a list of one or more lists.
L[i]	Subscript operator for L_i
ratexpand (expr) or expand(expr)	Expands expr by multiplying out products of sums and exponentiated sums, combining fractions over a common denominator, cancelling the greatest common divisor of the numerator and denominator, then splitting the numerator (if a sum) into its respective terms divided by the denominator.

Note:1. Press Shift+Enter for evaluation of commands and display of output.

2. Replace semicolon (;) by dollar ($) to suppress output of any input line and vice-versa.

3. Start each session with kill(all)$ or quit()$ to remove previously assigned values of all symbols

Definitions and Formulae:

Lagrange's Interpolation Polynomial: Let $(x_i, y_i), i = 0,1,2,3,\dots,n$ be $n+1$ data points of an unknown function $y = f(x)$. Here abscissa x_i, $i = 0,1,2,3,\dots,n$ are distinct and not necessarily be equally spaced. In Lagrange's Method, the unknown function $y = f(x)$ is approximated by a polynomial of degree n , called Lagrange's interpolation polynomial and is given by:

$$\boldsymbol{y = f(x) = \sum_{i=0}^{n} l_i(x) y_i}$$

where,

$$\boldsymbol{l_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)} = \frac{(x - x_0)}{(x_i - x_0)} \frac{(x - x_1)}{(x_i - x_1)} \cdots \frac{(x - x_{i-1})}{(x_i - x_{i-1})} \frac{(x - x_{i+1})}{(x_i - x_{i+1})} \cdots \frac{(x - x_n)}{(x_i - x_n)}}$$

Lagrange interpolation polynomial formula for $n = 3$ (for 4 points) is given below:

$$\boldsymbol{f(x) = \frac{(x - x_1)}{(x_0 - x_1)} \frac{(x - x_2)}{(x_0 - x_2)} \frac{(x - x_3)}{(x_0 - x_3)} y_0 + \frac{(x - x_0)}{(x_1 - x_0)} \frac{(x - x_2)}{(x_1 - x_2)} \frac{(x - x_3)}{(x_1 - x_3)} y_1}$$
$$\boldsymbol{+ \frac{(x - x_0)}{(x_2 - x_0)} \frac{(x - x_1)}{(x_2 - x_1)} \frac{(x - x_3)}{(x_2 - x_3)} y_2 + \frac{(x - x_0)}{(x_3 - x_0)} \frac{(x - x_1)}{(x_3 - x_1)} \frac{(x - x_2)}{(x_3 - x_2)} y_3}$$

Lagrange interpolation polynomial formula for $n = 4$ (for 5 points) is given below:

$$\boldsymbol{f(x) = \frac{(x - x_1)}{(x_0 - x_1)} \frac{(x - x_2)}{(x_0 - x_2)} \frac{(x - x_3)}{(x_0 - x_3)} \frac{(x - x_4)}{(x_0 - x_4)} y_0}$$
$$\boldsymbol{+ \frac{(x - x_0)}{(x_1 - x_0)} \frac{(x - x_2)}{(x_1 - x_2)} \frac{(x - x_3)}{(x_1 - x_3)} \frac{(x - x_4)}{(x_1 - x_4)} y_1}$$
$$\boldsymbol{+ \frac{(x - x_0)}{(x_2 - x_0)} \frac{(x - x_1)}{(x_2 - x_1)} \frac{(x - x_3)}{(x_2 - x_3)} \frac{(x - x_4)}{(x_2 - x_4)} y_2}$$
$$\boldsymbol{+ \frac{(x - x_0)}{(x_3 - x_0)} \frac{(x - x_1)}{(x_3 - x_1)} \frac{(x - x_2)}{(x_3 - x_2)} \frac{(x - x_4)}{(x_3 - x_4)} y_3}$$
$$\boldsymbol{+ \frac{(x - x_0)}{(x_4 - x_0)} \frac{(x - x_1)}{(x_4 - x_1)} \frac{(x - x_2)}{(x_4 - x_2)} \frac{(x - x_3)}{(x_4 - x_3)} y_4}$$

Program:

Program to find the Lagrange interpolation polynomial $y = f(x)$ for the given table of values:

x	x_0	x_1	x_2	x_3	x_4	x_5	x_6
y	y_0	y_1	y_2	y_3	y_4	y_5	y_6

and finding $f(x)$ at $x = a$

Program:

```
kill(all)$
load("interpol")$
X:[x0,x1,x2,x3,x4,x5,x6]$
Y:[y0,y1,y2,y3,y4,y5,y6]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrage Interpolation Polynomial is f(x)=",f)$
print("f(a)=",at(f,x=a))$
```

Program to find the Missing value(s) of the table by Lagrange Method:

x	x_0	x_1	x_2	x_3	x_4	x_5	x_6
y	y_0	y_1	$-$	y_3	y_4	$-$	y_6

Note: For writing X and Y, consider those points for which both (x_i, y_i) are known. Don't consider x_i for which y_i is missing.

Program:

```
kill(all)$
load("interpol")$
X:[x0,x1,x3,x4,x6]$
Y:[y0,y1,y3,y4,y6]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
printCf)$
print("Missing values of the table are")$
print("f(x2)=",at(f,x=x2))$
print("f(x5)=",at(f,x=x5))$
```

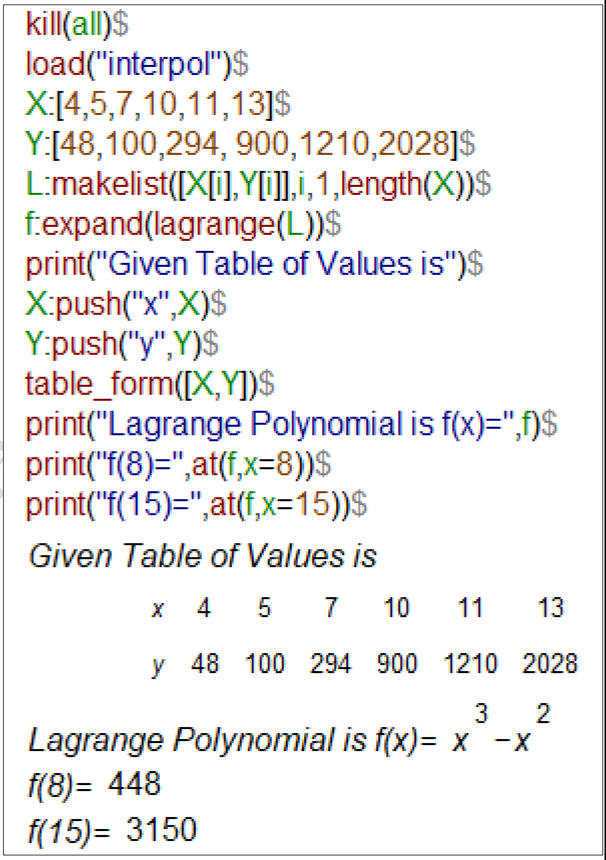
Worked Examples:

Problem 1. Write a program to find the Lagrange interpolation polynomial $y = f(x)$ for the given table of values. Also find $f(8)$ and $f(15)$.

x	4	5	7	10	11	13
y	48	100	294	900	1210	2028

Program:

```
kill(all)$
load("interpol")$
X:[4,5,7,10,11,13]$
Y:[48,100,294, 900,1210,2028]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrange Polynomial is f(x)=",f)$
print("f(8)=",at(f,x=8))$
print("f(15)=",at(f,x=15))$
```



```
kill(all)$
load("interpol")$
X:[4,5,7,10,11,13]$
Y:[48,100,294, 900,1210,2028]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrange Polynomial is f(x)=",f)$
print("f(8)=",at(f,x=8))$
print("f(15)=",at(f,x=15))$
```

Given Table of Values is

x	4	5	7	10	11	13
y	48	100	294	900	1210	2028

Lagrange Polynomial is f(x)= $x^3 - x^2$

f(8)= 448

f(15)= 3150

Output:

Given Table of Values is

x	5	6	7	8	9	10
y	196	394	686	1090	1624	2306

Lagrange Polynomial is $f(x) = x^3 - x^2$

$f(8) = 448$

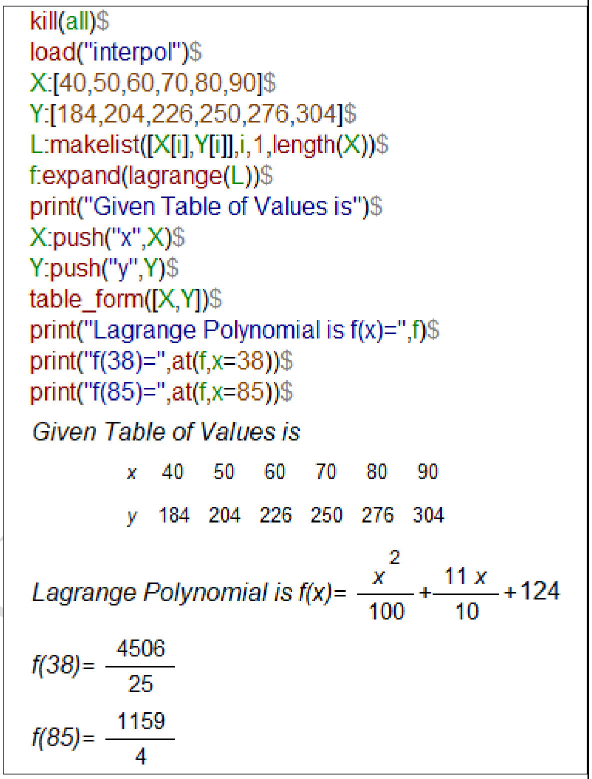
$f(15) = 3150$

Problem 2. Write a program to find the Lagrange interpolation polynomial $y = f(x)$ for the given table of values. Also find $f(38)$ and $f(85)$.

x	40	50	60	70	80	90
y	184	204	226	250	276	304

Program:

```
kill(all)$
load("interpol")$
X:[40,50,60,70,80,90]$
Y:[184,204,226,250,276,304]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrange Polynomial is f(x)=",f)$
print("f(38)=",at(f,x=38))$
print("f(85)=",at(f,x=85))$
```



Output:

Given Table of Values is

x	40	50	60	70	80	90
y	184	204	226	250	276	304

$$\text{Lagrange Polynomial is } f(x) = \frac{x^2}{100} + \frac{11x}{10} + 124$$

$$f(38) = \frac{4506}{25}$$

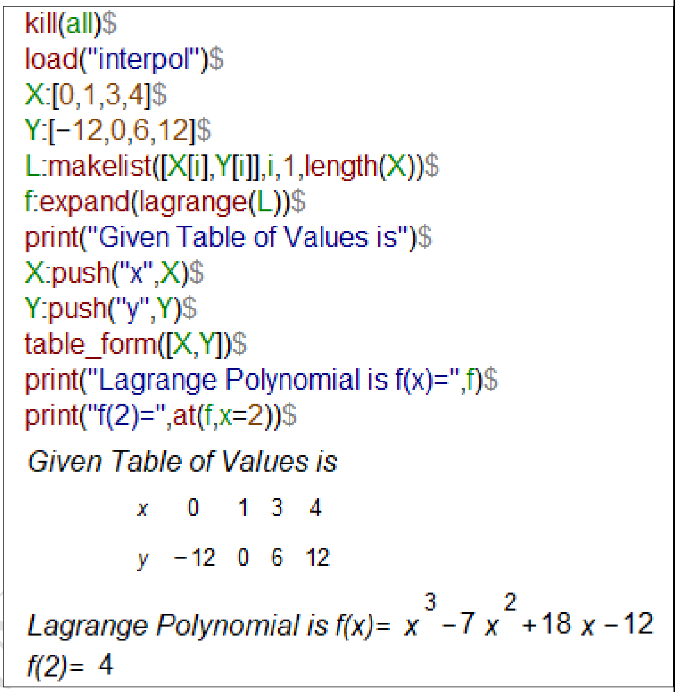
$$f(85) = \frac{1159}{4}$$

Problem 3. Write a program to find the Lagrange interpolation polynomial $y = f(x)$ for the given table of values. Also find $f(2)$.

x	0	1	3	4
y	-12	0	6	12

Program:

```
kill(all)$
load("interpol")$
X:[0,1,3,4]$
Y:[-12,0,6,12]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrange Polynomial is f(x)=",f)$
print("f(2)=",at(f,x=2))$
```



```
kill(all)$
load("interpol")$
X:[0,1,3,4]$
Y:[−12,0,6,12]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrange Polynomial is f(x)=",f)$
print("f(2)=",at(f,x=2))$
```

Given Table of Values is

x	0	1	3	4
y	-12	0	6	12

Lagrange Polynomial is f(x)= $x^3 - 7x^2 + 18x - 12$

f(2)= 4

Output:

Given Table of Values is

x	0	1	3	4
y	-12	0	6	12

Lagrange Polynomial is $f(x) = x^3 - 7x^2 + 18x - 12$

$f(2) = 4$

Problem 4. Write a program to find the missing values in the given table of values of an unknown function $y = f(x)$ by Lagrange Method.

x	4	5	6	7	8	9
y	72	–	146	192	–	302

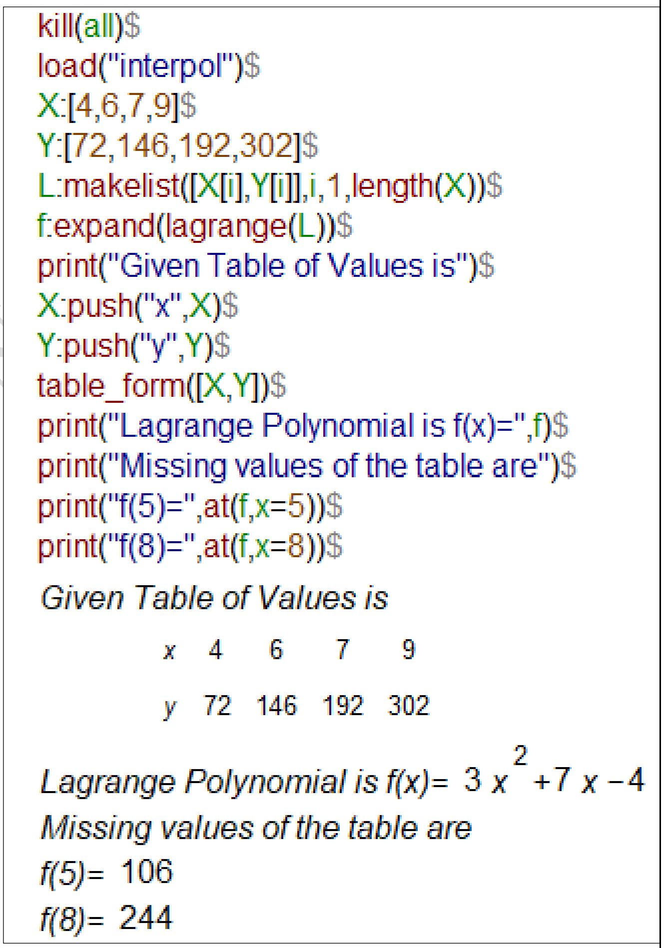
Here known values are

x	4	6	7	9
y	72	146	192	302

And Missing values are $f(5)$ and $f(8)$.

Program:

```
kill(all)$
load("interpol")$
X:[4,6,7,9]$
Y:[72,146,192,302]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrange Polynomial is f(x)=",f)$
print("Missing values of the table are")$
print("f(5)=",at(f,x=5))$
print("f(8)=",at(f,x=8))$
```



```
kill(all)$
load("interpol")$
X:[4,6,7,9]$
Y:[72,146,192,302]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrange Polynomial is f(x)=",f)$
print("Missing values of the table are")$
print("f(5)=",at(f,x=5))$
print("f(8)=",at(f,x=8))$
```

Given Table of Values is

x	4	6	7	9
y	72	146	192	302

Lagrange Polynomial is f(x)= $3x^2+7x-4$

Missing values of the table are

f(5)= 106

f(8)= 244

Output:

$Given\ Table\ of\ Values\ is$

x	4	6	7	9
y	72	146	192	302

$Lagrange\ Polynomial\ is\ f(x) = 3x^2 + 7x - 4$

$Missing\ values\ of\ the\ table\ are$

$f(5) = 106$

$f(8) = 224$

Problem 5. Write a program to find the missing value in the given table of values of an unknown function $y = f(x)$ by Lagrange Method.

x	0	1	2	3	4
y	148	192	241	–	374

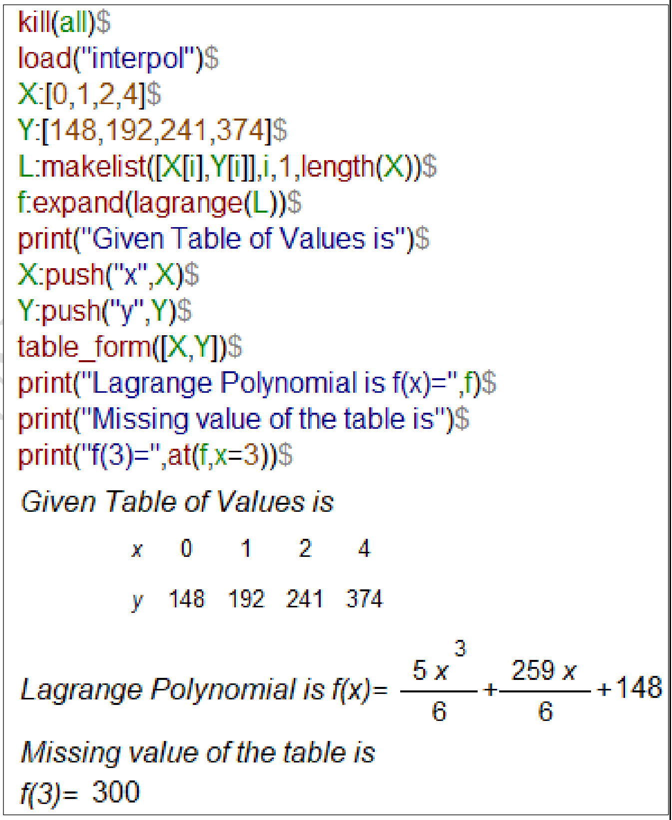
Here known values are

x	0	1	2	4
y	148	192	241	374

and Missing value is $f(3)$.

Program:

```
kill(all)$
load("interpol")$
X:[0,1,2,4]$
Y:[148,192,241,374]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrange Polynomial is f(x)=",f)$
print("Missing value of the table is")$
print("f(3)=",at(f,x=3))$
```



```
kill(all)$
load("interpol")$
X:[0,1,2,4]$
Y:[148,192,241,374]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrange Polynomial is f(x)=",f)$
print("Missing value of the table is")$
print("f(3)=",at(f,x=3))$
```

Given Table of Values is

x	0	1	2	4
y	148	192	241	374

Lagrange Polynomial is f(x)= $\frac{5x^3}{6} + \frac{259x}{6} + 148$

Missing value of the table is

f(3)= 300

Output:

$Given\ Table\ of\ Values\ is$

x	0	1	2	4
y	148	192	241	374

$$Lagrange\ Polynomial\ is\ f(x) = \frac{5x^3}{6} + \frac{259x}{6} + 148$$

$Missing\ value\ of\ the\ table\ is$

$f(3) = 300$

Problem 6. Write a program to find the missing value in the given table of values of an unknown function $y = f(x)$ by Lagrange Method.

x	19	20	21	22	23
y	91	100.25	110	–	131

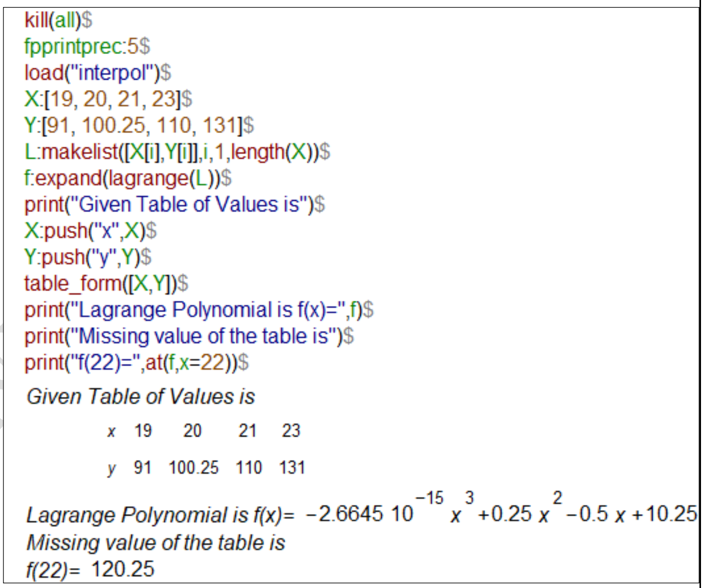
Here known values are

x	19	20	21	23
y	91	100.25	110	131

and Missing value is $f(22)$.

Program:

```
kill(all)$
fpprintprec:5$
load("interpol")$
X:[19, 20, 21, 23]$
Y:[91, 100.25, 110, 131]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrange Polynomial is f(x)=",f)$
print("Missing value of the table is")$
print("f(22)=",at(f,x=22))$
```



```
kill(all)$
fpprintprec:5$
load("interpol")$
X:[19, 20, 21, 23]$
Y:[91, 100.25, 110, 131]$
L:makelist([X[i],Y[i]],i,1,length(X))$
f:expand(lagrange(L))$
print("Given Table of Values is")$
X:push("x",X)$
Y:push("y",Y)$
table_form([X,Y])$
print("Lagrange Polynomial is f(x)=",f)$
print("Missing value of the table is")$
print("f(22)=",at(f,x=22))$
```

Given Table of Values is

x	19	20	21	23
y	91	100.25	110	131

Lagrange Polynomial is f(x)= $-2.6645\ 10^{-15}\ x^3 + 0.25\ x^2 - 0.5\ x + 10.25$
Missing value of the table is
f(22)= 120.25

Output:

$$Given\ Table\ of\ Values\ is$$

x	19	20	21	23
y	91	100.25	110	131

$$Lagrange\ Polynomial\ is\ f(x) = -2.664510^{-15}x^3 + 0.25x^2 - 0.5x + 10.25$$

$$Missing\ value\ of\ the\ table\ is$$

$$f(22) = 120.25$$

Exercise:

1. Write a program to find the Lagrange interpolation polynomial $y = f(x)$ for the given table of values. Also find $f(6)$.

x	-2	1	3	7	8
y	10	4	40	424	620

(Answer: $f(x) = x^3 + 2x^2 - 3x + 4, f(6) = 274$)

2. Write a program to find the Lagrange interpolation polynomial $y = f(x)$ for the given table of values. Also find $f(10)$.

x	5	6	9	11
y	12	13	14	16

(Answer: $f(x) = \frac{x^3}{20} - \frac{7x^2}{6} + \frac{557x}{60} - \frac{23}{2}, f(10) = \frac{44}{3}$)

3. Write a program to find the missing values in the given table of values of an unknown function $y = f(x)$ by Lagrange Method.

x	45	50	55	60	65
y	3	$-$	2	$-$	2.4

(Answer: $f(x) = 0.007x^2 - 0.8x + 24.82, f(50) = 2.325, f(60) = 2.025$)

4. Write a program to find the missing values in the given table of values of an unknown function $y = f(x)$ by Lagrange Method.

x	0	1	2	3	4	5
y	0	$-$	8	15	$-$	35

(Answer: $f(x) = x^2 + 2x, f(1) = 3, f(4) = 24$)

5. Write a program to find the missing value in the given table of values of an unknown function $y = f(x)$ by Lagrange Method.

x	0	1	2	3	4	5
y	-1	3	19	53	$-$	199

(Answer: $f(x) = x^3 + 3x^2 - 1, f(4) = 111$)

Thank You

Lakkanna E N,
Assistant Professor in Mathematics,
Mobile:8050153462
Email: lakkannaen.83@gmail.com

This is an effort to learn Maxima

(Please ignore typing errors, if any)

(Suggestions to improve this manual are welcome)