Inna Williams

########################################################################

## Section 2.7

************************************************************************

**1. Implement Newton's Method with appropriate starting points to find all solutions. Check with Exercise 3 to make sure your answers are correct.**

************************************************************************

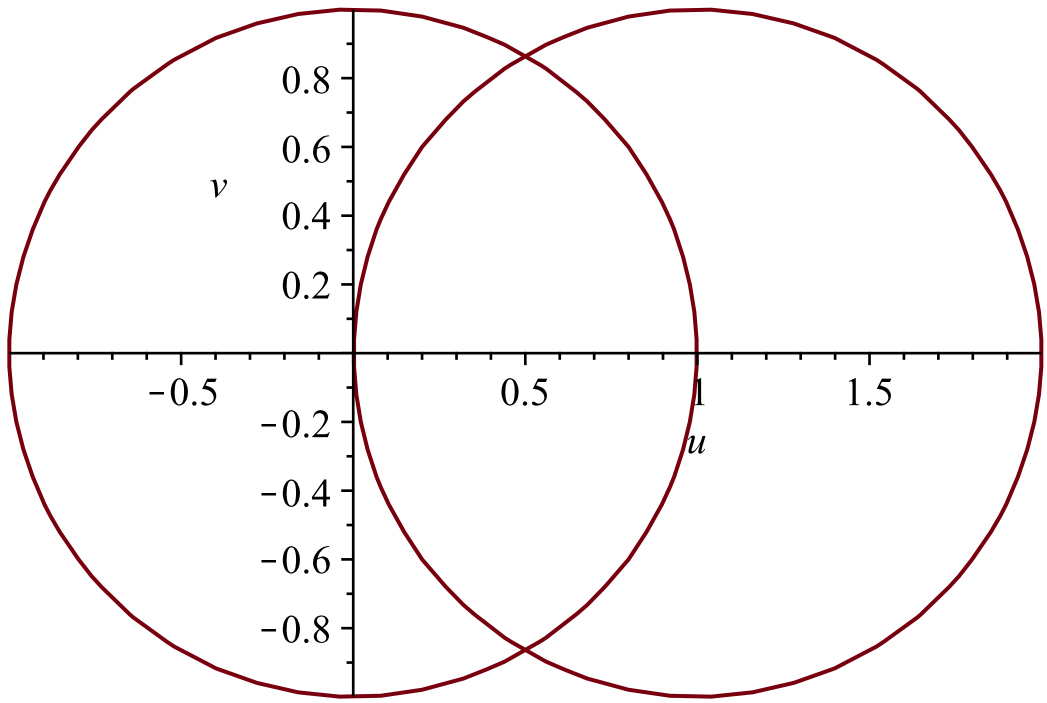
**a)**

************************************************************************

$$u^2 + v^2 = 1$$

$$(u-1)^2 + v^2 = 1$$

```
> restart;
> with(plots, implicitplot) : with(plots) : with(LinearAlgebra) : with(VectorCalculus) :
> newtonMD := proc(X0, Y0, TOL, N, x, y)
        local i, s, x0, y0, sol, A;
        x0 := X0; y0 := Y0; sol := ⟨x0, y0⟩;
        i := 1;
        while i ≤ N do
            A := Jacobian(F, [x, y] = [x0, y0]);
            if Determinant(A) = 0 then
                printf("Non-invertible Jacobian. Method failed");
                break;
            else
            s := LinearSolve(A, subs([x = x0, y = y0], -F));
            sol := sol + s;
            x0 := DotProduct(sol, ⟨1, 0⟩); y0 := DotProduct(sol, ⟨0, 1⟩);
            printf("Step %d: x=%.8g, y=%.8g \n", i, x0, y0);
            if evalf(Norm(s)) < TOL then
               printf("Number of iterations needed: %d", i);return( );
               break;
            end if;
            i := i + 1;
            end if;
        end do;
        printf("The method failed after %d iterations.\n", N);
        return( );
    end proc:
> implicitplot([(u-1)^2 + v^2 - 1 = 0, u^2 + v^2 = 1], u = -1 ..2, v = -1 ..1, scaling = constrained);
```




> $F := \langle u^2+v^2-1,\ (u-1)^2+v^2-1\rangle;$

$$F := \left(u^2+v^2-1\right)e_x + \left((u-1)^2+v^2-1\right)e_y \tag{1}$$

> $newtonMD\left(1.0, 1.0, 0.5\cdot 10^{-6}, 100, u, v\right)$

```
Step 1: x=0.5, y=1
Step 2: x=0.5, y=0.875
Step 3: x=0.5, y=0.86607143
Step 4: x=0.5, y=0.86602541
Step 5: x=0.5, y=0.8660254
Number of iterations needed: 5
```

**Answer: Solution to system of equations are:**

**x(u,v) =(0.5, 0.8660254)**

**corresponds to the answer of 3a in not computer problems**

> $u := evalf\left(\frac{1}{2}\right); v := evalf\left(\frac{\left(3^{\frac{1}{2}}\right)}{2}\right)$

$$u := 0.5000000000$$

$$v := 0.8660254040 \tag{2}$$

########################################################################
**3. Use Newton's Method to find the two solutions of the system**
###############################################################################

```
> restart;
> with(plots, implicitplot) : with(plots) : with(LinearAlgebra) : with(VectorCalculus) :
> newtonMD :=proc(X0, Y0, TOL, N, x, y)
        local i, s, x0, y0, sol, A;
        x0 := X0; y0 := Y0; sol := ⟨x0, y0⟩;
        i := 1;
        while i ≤ N do
           A := Jacobian(F, [x, y] = [x0, y0]);
           if Determinant(A) = 0 then
                 printf("Non-invertible Jacobian. Method failed");
                 break;
            else
            s := LinearSolve(A, subs([x = x0, y = y0], -F));
            sol := sol + s;
            x0 := DotProduct(sol, ⟨1, 0⟩); y0 := DotProduct(sol, ⟨0, 1⟩);
            printf("Step %d: x=%.8g, y=%.8g \n", i, x0, y0);
            if evalf(Norm(s)) < TOL then
               printf("Number of iterations needed: %d", i);return( );
               break;
             end if;
           i := i + 1;
           end if;
        end do;
        printf("The method failed after %d iterations.\n", N);
        return( );
   end proc:
>
```

> $F := \langle u^3 - v^3 + u,\ u^2 + v^2 - 1\rangle;$

$$F := \left(u^3 - v^3 + u\right)e_x + \left(u^2 + v^2 - 1\right)e_y \qquad (3)$$

> $newtonMD\left(1.0, 1.0, 0.5\cdot 10^{-6}, 100, u, v\right)$

```
Step 1: x=0.64285714, y=0.85714286
Step 2: x=0.52237409, y=0.86119563
Step 3: x=0.5081777, y=0.86136925
Step 4: x=0.50799203, y=0.86136179
Step 5: x=0.507992, y=0.86136179
Number of iterations needed: 5
```

> $newtonMD\left(-1, -1, 0.5\cdot 10^{-6}, 100, u, v\right)$

```
Step 1: x=-0.64285714, y=-0.85714286
Step 2: x=-0.52237409, y=-0.86119562
Step 3: x=-0.5081777, y=-0.86136925
Step 4: x=-0.50799203, y=-0.86136179
Step 5: x=-0.507992, y=-0.86136179
Number of iterations needed: 5
```

**Answer:**

**Solutions to system of equations are:**
**using initial value x0=(1,1) : u=0.507992, v=0.86136179**
**using initial value x0=(-1,-1) : u=-0.507992, v=-0.86136179**

\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#

## Section 13.1

\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#

**1.** ***Plot the function*** $y = f\cdot(x)$**, and** ***find a length − one starting interval on which f is unimodal around each relative minimum. Then apply Golden Section Search*** **to** ***locate each of the function's relative minima*** **to** ***within five correct digits.***

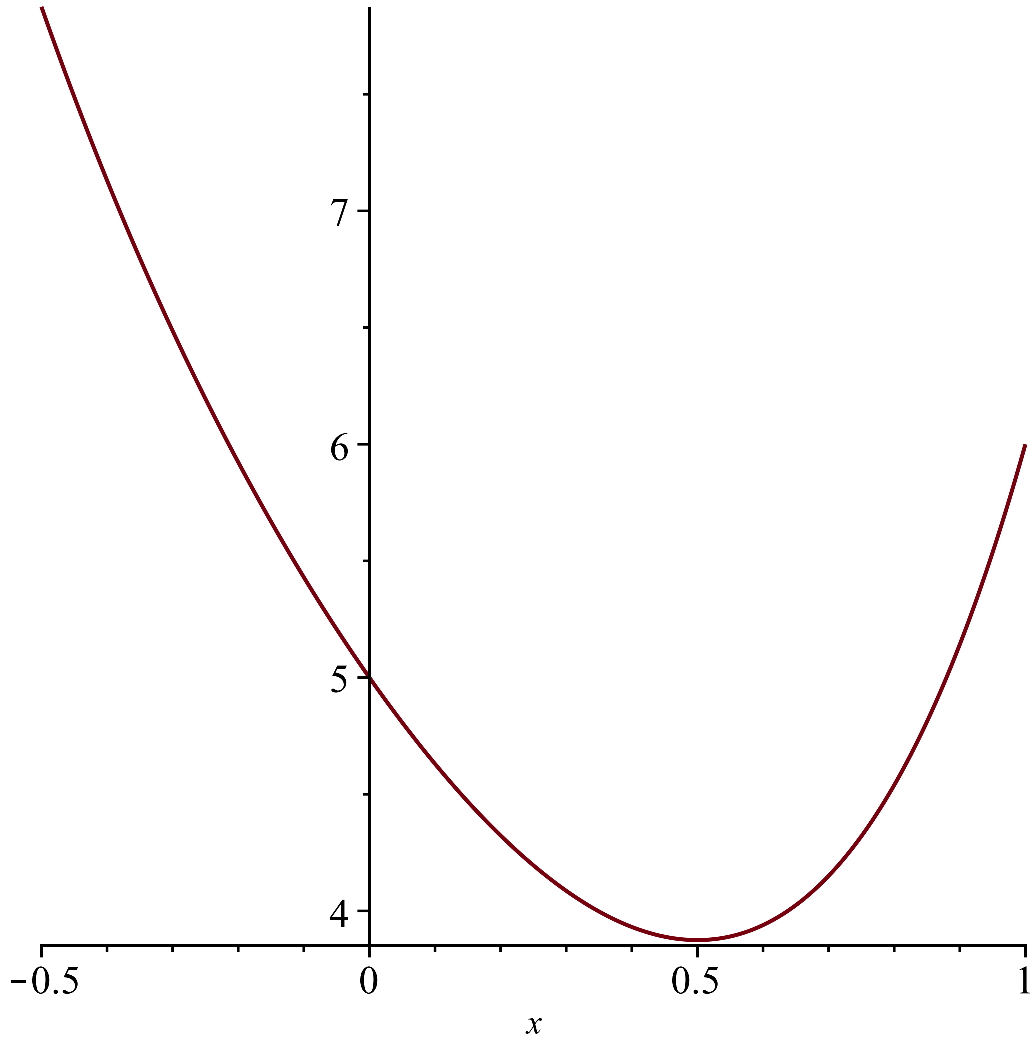
```
> restart; with(plots) :
  GoldenSearch ≔proc(A, B, TOL, N)
          local i, g, a, b, p; a ≔ A; b ≔ B;
          i ≔ 0;
          g ≔ evalf(((√5 − 1))/2);
          while i ≤ N do
             p ≔ a + (b-a) / 2 :
             if  (b-a) / 2 < TOL then
                 printf("xmin=%.8g, ymin=%.8g\n", p, f(p));
                 printf("Number of iterations needed: %d", i);return( );
             break;
             end if;
             if f(a + (1-g) * (b-a)) < f(a + g * (b-a))  then
                 b ≔ a + g * (b-a);
              else
                 a ≔ a + (1-g) * (b-a);
              end if;
              i ≔ i + 1;
          end do;
          printf("The method failed after %d iterations.\n", N);
   end proc:
```

***a***) $f(x) = 2x^4 + 3x^2 - 4x + 5$

```
> f := x→2·x^4 + 3·x^2 − 4 x + 5
```

$$f := x \mapsto 2x^4 + 3x^2 - 4x + 5 \quad (4)$$

```
> p1 := plot(f(x), x = -0.5 .. 1.0): display(p1)
```




Plot shows that interval [0,1] contain relative minimum
According to the theorem 13.2 the number k of the Golden Search steps needed satisfies

$\frac{g^k(1-0)}{2} < 0.5*10^{-5}$ *where* $g = \frac{\left(5^{\frac{1}{2}}-1\right)}{2}$

$$\left(\frac{\left(\frac{\left(5^{\frac{1}{2}}-1\right)}{2}\right)^k \cdot (1-0)}{2}\right) < 0.5*10^{-5}$$

k $<= \frac{\log10\left(10^{-5}\right)}{\log10\left(\frac{\left(5^{\frac{1}{2}}-1\right)}{2}\right)}$

> $k := evalf\left(\frac{\log10(10^{-5})}{\log10\left(\frac{\left(5^{\frac{1}{2}}-1\right)}{2}\right)}\right)$

$$k := 23.92485978 \tag{5}$$

Number of steps needed k >= 24

> $GoldenSearch(0.0, 1.0, 0.5\cdot 10^{-5}, 100)$

```
xmin=0.50000964, ymin=3.875
Number of iterations needed: 24
```

**Answer: Relative minima= [x = 0.5,  y = 3.875]**
**Number of iterations correspond to the theoretical value >=24**
****************************************************************************

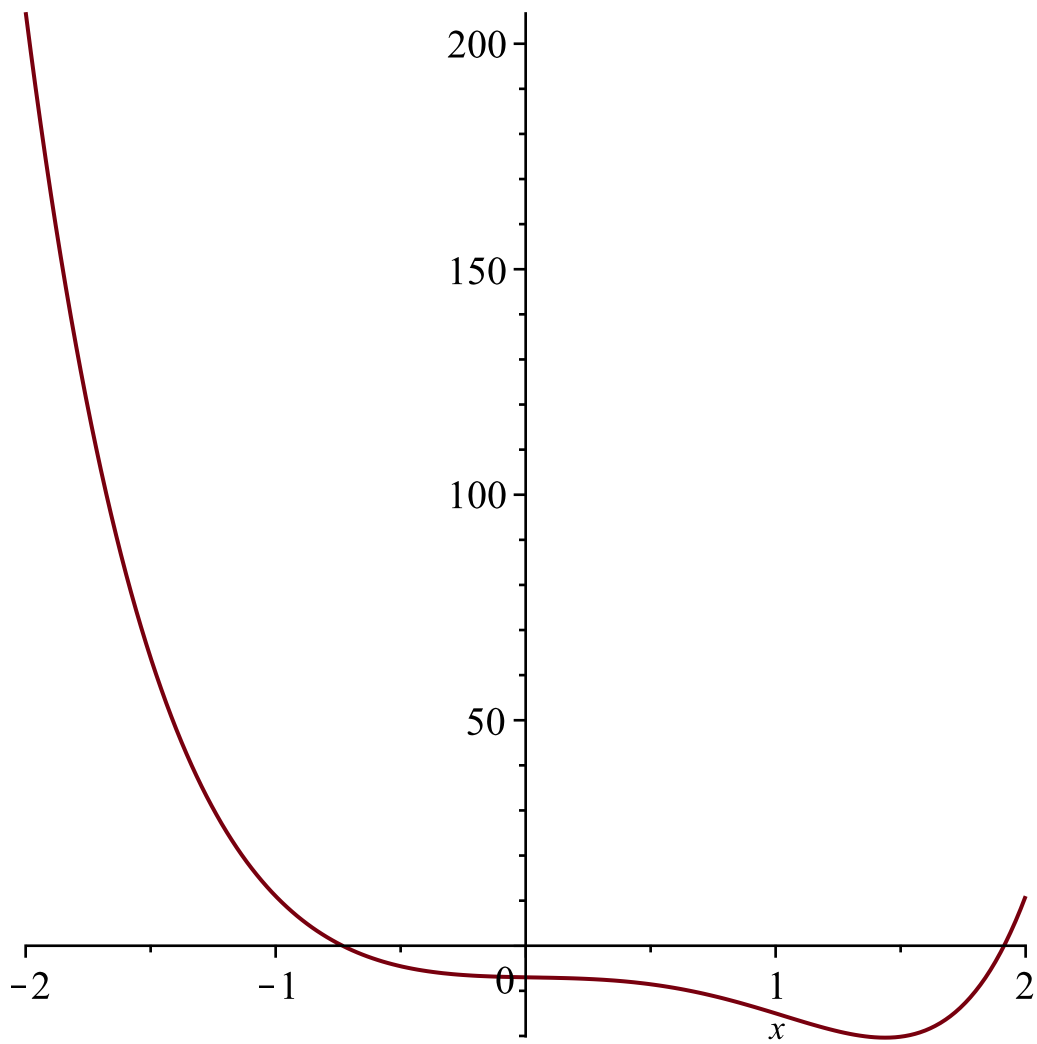
**d)** $\boldsymbol{f(x) = x^6 + 3x^4 - 12x^3 + x^2 - x - 7}$

$$2x^4 + 3x^2 - 4x + 5 = x^6 + 3x^4 - 12x^3 + x^2 - x - 7 \tag{6}$$

> $f := x \rightarrow x^6 + 3x^4 - 12x^3 + x^2 - x - 7$

$$f := x \mapsto x^6 + 3x^4 - 12x^3 + x^2 - x - 7 \tag{7}$$

> $p1 := plot(f(x), x = -2\,..\,2) : display(p1)$



Plot shows that interval [1,2] contain relative minimum

$$\frac{g^k(1-0)}{2} < 0.5*10^{-5} \quad where \ \ g = \frac{\left(5^{\frac{1}{2}} - 1\right)}{2}$$

$$\left( \frac{\left( \frac{\left(5^{\frac{1}{2}} - 1\right)}{2} \right)^k \cdot (2-1)}{2} \right) < 0.5*10^{-5}$$

$$\left( \left( \frac{\left(5^{\frac{1}{2}} - 1\right)}{2} \right)^k \right) < 10^{-5}$$

$$k <= \frac{\log10(10^{-5})}{\log10\left( \frac{\left(5^{\frac{1}{2}} - 1\right)}{2} \right)}$$

> $k := evalf\left( \frac{\log10(10^{-5})}{\log10\left( \frac{\left(5^{\frac{1}{2}} - 1\right)}{2} \right)} \right)$

$$k := 23.92485978 \tag{8}$$

> $GoldenSearch(1, 2, 0.5 \cdot 10^{-5}, 100)$

```
xmin=1.4379188, ymin=-20.382879
Number of iterations needed: 24
```

**Answer: Relative minima= [x = 1.4379188,  y = -20.382879]**
**Number of iterations correspond to the theoretical value >=24**

**************************************************************

**5. Use the Nelder–Mead Method to find the minimum of $f(x,y) = \exp(-x^2y^2) + (x-1)^2 + (y-1)^2$ Try various initial conditions, and compare answers. How many correct digits can you obtain by using this method?**

> $f := (x, y) \rightarrow evalf(\operatorname{abs}(\exp(-x^2y^2) + (x-1)^2 + (y-1)^2))$

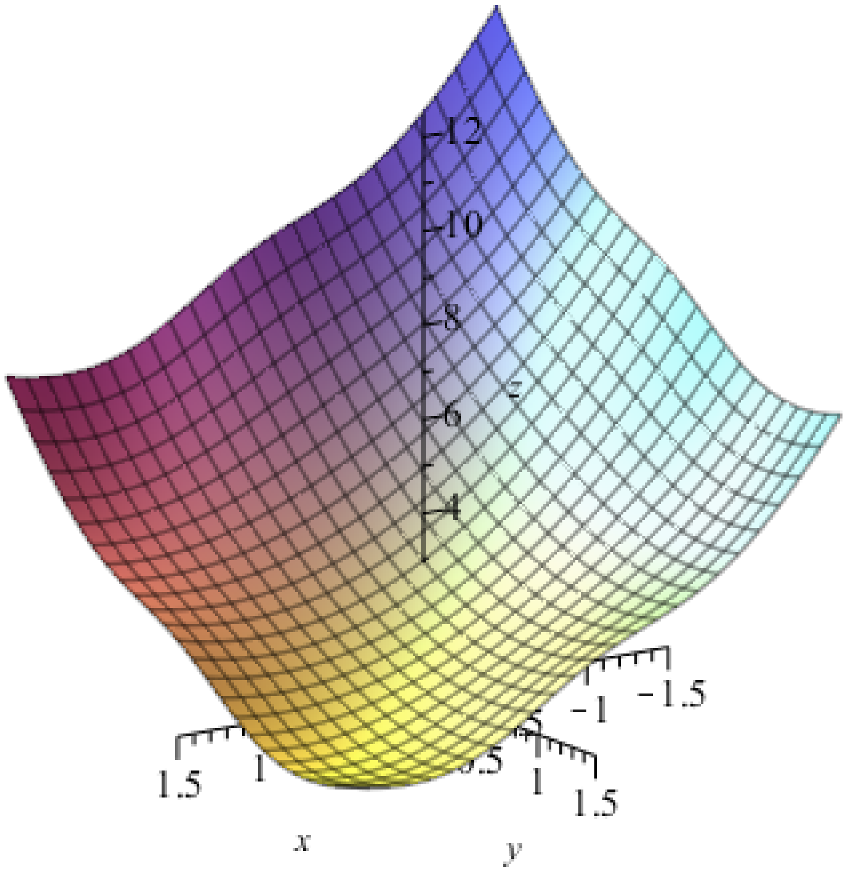
$$f := (x, y) \mapsto evalf\left( \left| e^{-x^2 y^2} + (x-1)^2 + (y-1)^2 \right| \right) \tag{9}$$

```
> xmin:= -1.5; xmax:= 1.5; ymin:= -1.5; ymax:= 1.5;
```

$$xmin := -1.5$$
$$xmax := 1.5$$
$$ymin := -1.5$$
$$ymax := 1.5 \tag{10}$$

```
> with(plots):
> plot3d(f(x,y),x=xmin..xmax,y=ymin..ymax,axes=normal,labels=[x,y,
```

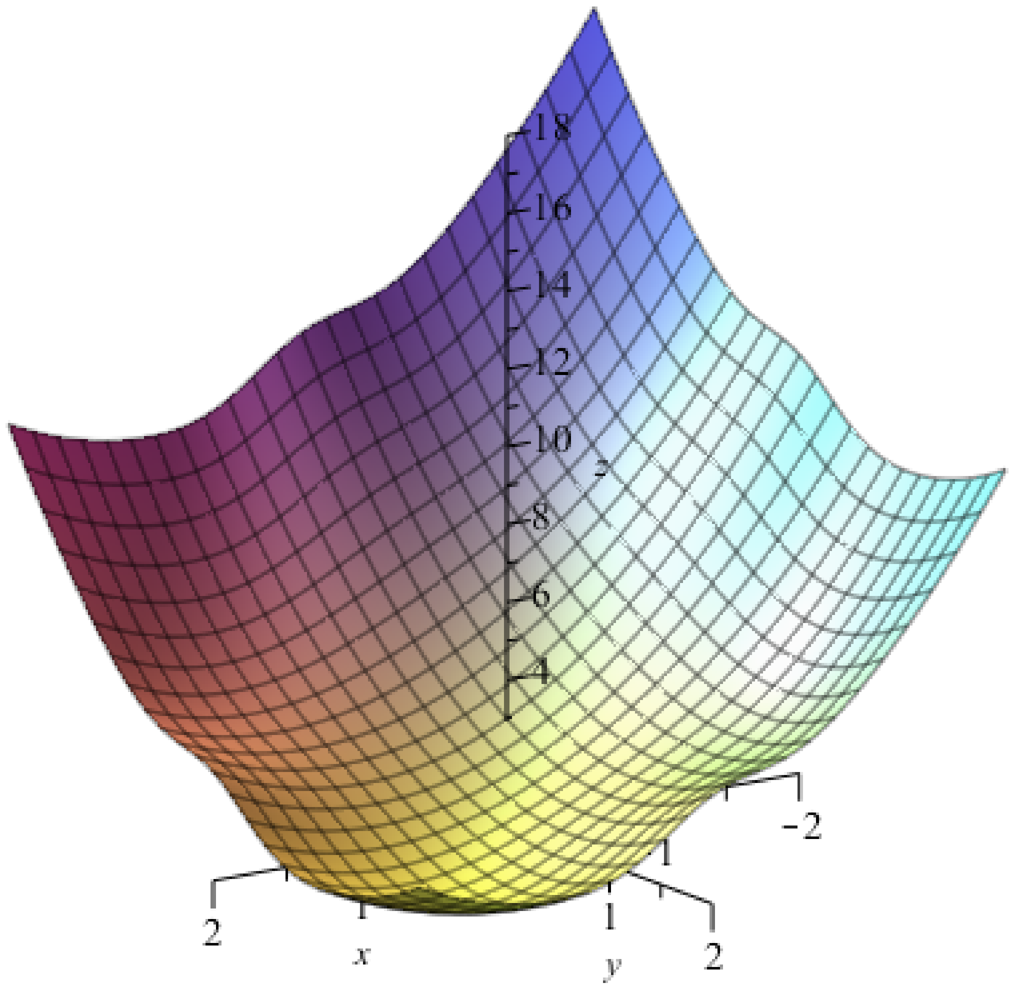
```
z]);
```



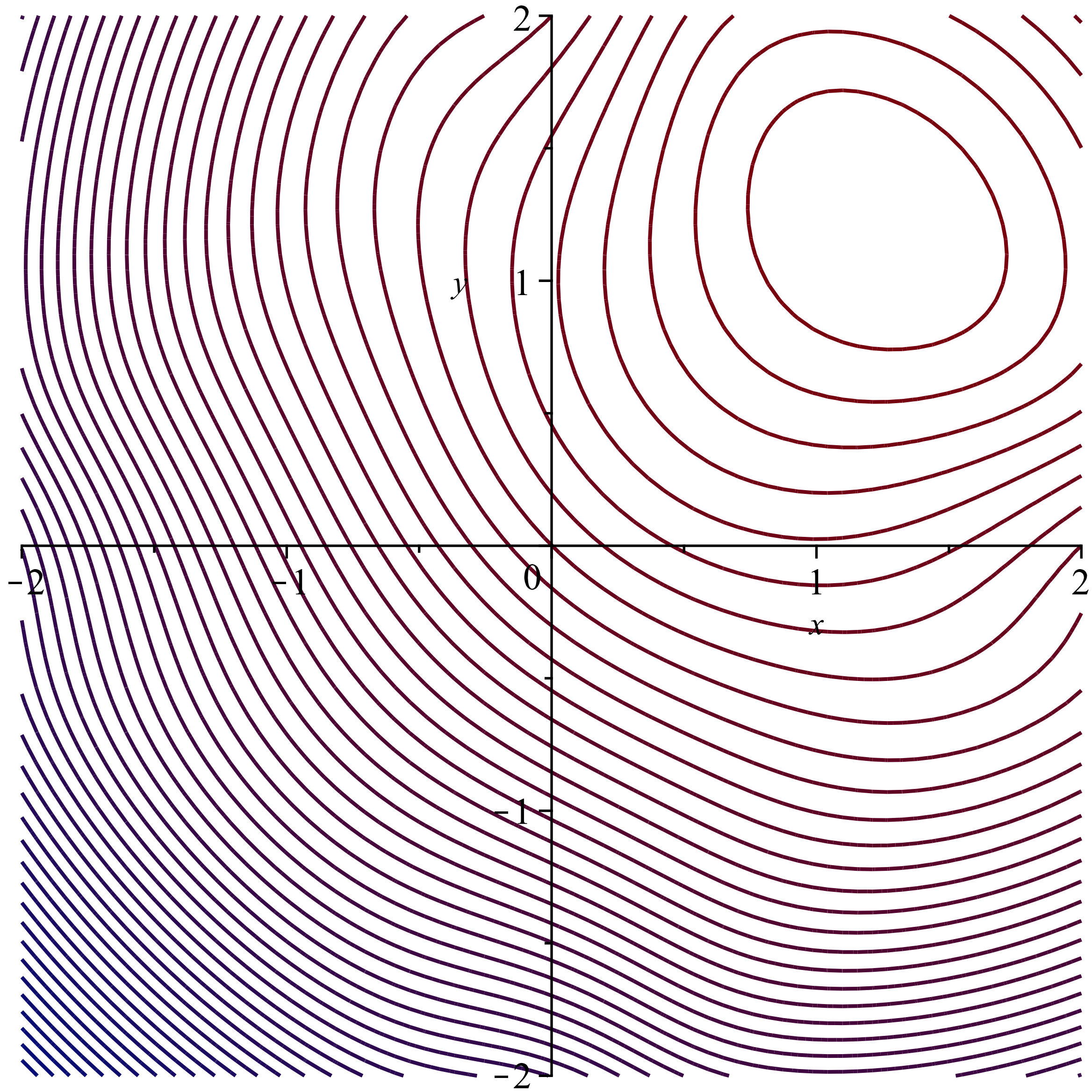

```
> xmin:= -2; xmax:= 2; ymin:= -2; ymax:= 2;
```

$$xmin := -2$$
$$xmax := 2$$
$$ymin := -2$$
$$ymax := 2 \qquad (1.1)$$

```
> with(plots):
> plot3d(f(x,y),x=xmin..xmax,y=ymin..ymax,axes=normal,labels=[x,
  y,z]);
```




```
> contourplot(f(x,y),x=xmin..xmax,y=ymin..ymax,scaling=
  constrained,contours=50, numpoints=5000);
```




Enter initial simplex, stopping tolerance

```
> y1:= [1.5, 0]; y2:= [2, 0]; y3:=[2,0.5]; #define initial
  simplex.  Need not be ordered
```

$$y1 := [1.5, 0]$$

$$y2 := [2, 0]$$

$$y3 := [2, 0.5] \quad \textbf{(2.1)}$$

```
> epsilon := 1.0 * 10^(-5); #stopping tolerance (see method
  summary: Termination)
```

$$\epsilon := 0.00001000000000 \quad \textbf{(2.2)}$$

Create function acting on points

```
> ff:=proc(y1)
      f(y1[1],y1[2]);
  end:
```

Procedure for ordering points by objective function values

```
> order3:=proc(g,y1,y2,y3)  #returns list of the 3 points in
  order of increasing g
  local x1, x2, x3,f1, f2, f3:
    f1:=g(y1);f2:=g(y2);f3:=g(y3);
    if (f1 < f2) then
      if (f2 < f3) then
        RETURN([y1, y2,y3]);
      else
        if (f1 < f3) then  #order is f1, f3, f2
          RETURN([y1, y3, y2]);
        else #order is is f3, f1, f2
          RETURN([y3, y1, y2]);
        end if:
     end if:
  else
     if (f1 < f3) then #order is f2, f1, f3
       RETURN([y2, y1, y3]);
     else
       if (f3 < f2) then #order is f3, f2, f1
         RETURN([y3, y2, y1]);
       else #order is f2, f3, f1
         RETURN([y2, y3, y1]);
       end if:
     end if:
  end if:
  end: #end procedure
```

Create first simplex

```
> xlist:=order3(ff,y1, y2, y3):
  x1:=xlist[1]: x2:=xlist[2]: x3:=xlist[3]:
```

Procedure for deciding which action to perform on simplex

```
> nm:=proc(ff,y1,y2,y3,ep)   #takes func., ordered simplex, tol.,
  outputs lambda only (or 'shrink', or 'STOP')
    local x, deltax, fy1, fy2, fy3, fminus, fzero, fplus, fone,
  ftwo:
    ######
    x := (y1+y2)/2.0:
    deltax := x - y3:
    fy1 := ff(y1):
    fy2 := ff(y2):
    fy3 := ff(y3):
    fminus := ff(x + (-1/2)*deltax):
    fzero := ff(x):
    fplus := ff(x + (1/2)*deltax):
    fone := ff(x + deltax):
    ftwo := ff(x + 2*deltax):
    ######
    if (max(abs(fzero - fy1), abs(fzero - fy3)) < ep) then
      return 'STOP':
    elif (fone < fy1) then
      if (ftwo < fone) then
        return 2;
      else
```

```
        return 1;
      end if:
    elif (fone < fy2) then
      return 1;
    elif (fone < fy3) then #try contraction w/ lambda = +1/2
      if (fplus < fy3) then
        return 0.5:
      else
        return 'shrink':
      end if:
    else #try contraction w/lambda = -1/2
      if (fminus < fy3) then
        return -0.5:
      else
        return 'shrink':
      end if:
    end if:
  end:   #end procedure
```

Execution of algorithm

```
> simplexlist[0]:=[x1,x2,x3]:
> lambda:=0:
> N:=0: #  # of iterations
> for i from 0 while ((i < 50) and not(lambda = 'STOP')) do
    N:=i:   #note the i<50 above.  Otherwise roundoff error can
  cause infinite loop.
    lambda:=nm(ff, x1, x2, x3, epsilon):
    lambdalist[i]:=lambda:
    if (lambda = 'STOP') then
      actionlist[i]:="STOP":
      x := (x1 + x2)/2.0:
      if (ff(x) < ff(x1)) then
        lastpoint:=x:
        minf := ff(x):
      else
        lastpoint:= x1:
        minf := ff(x1):
      end if:
    elif (lambda = 'shrink') then
      actionlist[i]="Shrink":
      x2:=(x1+x2)/2.0:
      x3:=(x1+x3)/2.0:
      newsimplex := order3(ff, x1, x2, x3):
      x1:=newsimplex[1]:
      x2:=newsimplex[2]:
      x3:=newsimplex[3]:
      simplexlist[i+1]:=[x1,x2,x3]:
    else
      if (lambda = 1) then
        actionlist[i]:="Reflect":
      elif (lambda = 1/2) then
        actionlist[i]:="Outside Contraction":
      else
        actionlist[i]:="Inside Contraction":
      end if:
      x := (x1 + x2)/2.0:
```

```
      deltax := x - x3:
      x3 := x + lambda*deltax:
      newsimplex := order3(ff, x1, x2, x3):
      x1:=newsimplex[1]:
      x2:=newsimplex[2]:
      x3:=newsimplex[3]:
      simplexlist[i+1]:=[x1,x2,x3]:
    end if:
  end do:
```

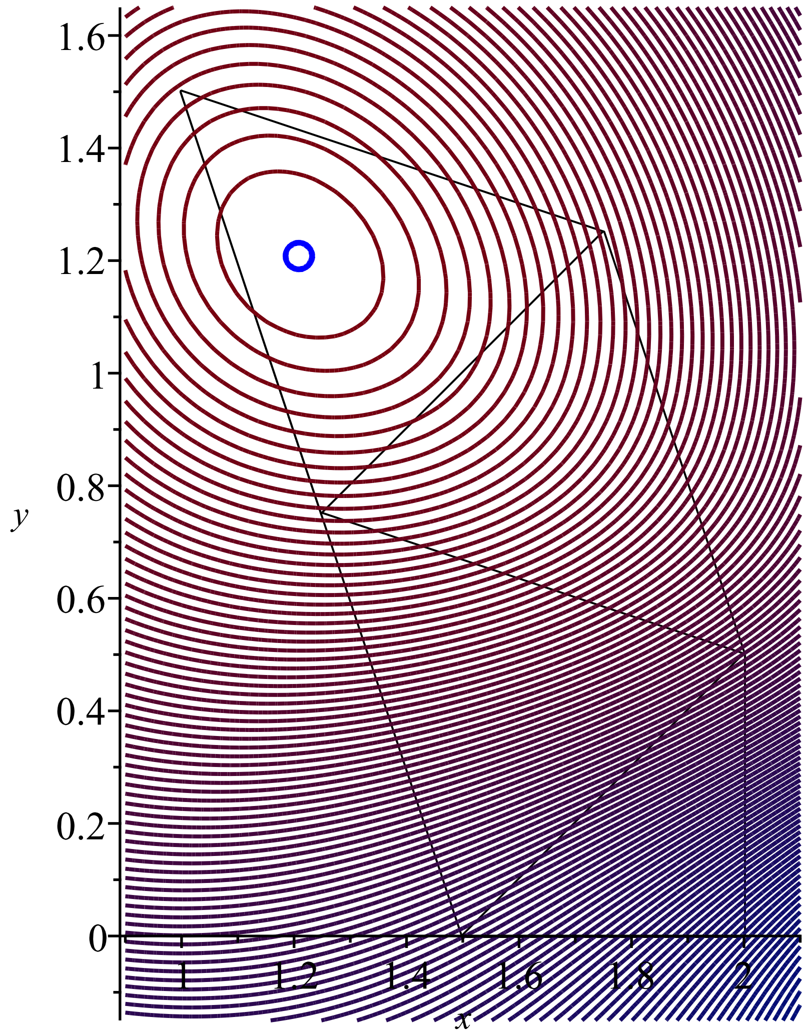
Print list of simplices, actions

```
> for i from 0 while (i < N) do
  printf("Simplex Number %d: (%f,%f),(%f,%f),(%f,%f) %s\n", i,
  simplexlist[i][1][1],simplexlist[i][1][2],
  simplexlist[i][2][1],simplexlist[i][2][2],
  simplexlist[i][3][1],simplexlist[i][3][2],
  actionlist[i]);
  end do:
Simplex Number 0: (2.000000,0.500000),(1.500000,0.000000),
(2.000000,0.000000) Inside Contraction
Simplex Number 1: (1.250000,0.750000),(2.000000,0.500000),
(1.500000,0.000000) Reflect
Simplex Number 2: (1.250000,0.750000),(1.750000,1.250000),
(2.000000,0.500000) Reflect
Simplex Number 3: (1.000000,1.500000),(1.250000,0.750000),
(1.750000,1.250000) Inside Contraction
Simplex Number 4: (1.437500,1.187500),(1.000000,1.500000),
(1.250000,0.750000) Inside Contraction
Simplex Number 5: (1.234375,1.046875),(1.437500,1.187500),
(1.000000,1.500000) Inside Contraction
Simplex Number 6: (1.167969,1.308594),(1.234375,1.046875),
(1.437500,1.187500) Inside Contraction
Simplex Number 7: (1.167969,1.308594),(1.319336,1.182617),
(1.234375,1.046875) Inside Contraction
Simplex Number 8: (1.239014,1.146240),(1.167969,1.308594),
(1.319336,1.182617) Inside Contraction
Simplex Number 9: (1.261414,1.205017),(1.239014,1.146240),
(1.167969,1.308594) Inside Contraction
Simplex Number 10: (1.209091,1.242111),(1.261414,1.205017),
(1.239014,1.146240) Inside Contraction
Simplex Number 11: (1.237133,1.184902),(1.209091,1.242111),
(1.261414,1.205017) Reflect
Simplex Number 12: (1.184811,1.221996),(1.237133,1.184902),
(1.209091,1.242111) Inside Contraction
Simplex Number 13: (1.210032,1.222780),(1.184811,1.221996),
(1.237133,1.184902) Inside Contraction
Simplex Number 14: (1.217277,1.203645),(1.210032,1.222780),
(1.184811,1.221996) Inside Contraction
Simplex Number 15: (1.217277,1.203645),(1.199232,1.217605),
(1.210032,1.222780) Reflect
Simplex Number 16: (1.217277,1.203645),(1.206478,1.198470),
(1.199232,1.217605) Inside Contraction
Simplex Number 17: (1.205555,1.209331),(1.217277,1.203645),
(1.206478,1.198470) Outside Contraction
Simplex Number 18: (1.205555,1.209331),(1.213885,1.210497),
(1.217277,1.203645) Outside Contraction
```

```
Simplex Number 19: (1.205555,1.209331),(1.205941,1.213049),
(1.213885,1.210497) Inside Contraction
Simplex Number 20: (1.209817,1.210844),(1.205555,1.209331),
(1.205941,1.213049) Reflect
Simplex Number 21: (1.209430,1.207126),(1.209817,1.210844),
(1.205555,1.209331) Inside Contraction
```

Draw simplices

```
> with(plottools):
> for i from 0 while (i < N+1) do
    triangles[i] := polygon([simplexlist[i][1], simplexlist[i]
  [2],  simplexlist[i][3]], color=white,labels=[x,y]):
  end do:
> xcoords1:= seq(simplexlist[i][1][1], i = 0..N):  #define
  plotting window
  xcoords2:= seq(simplexlist[i][2][1], i = 0..N):
  xcoords3:= seq(simplexlist[i][3][1], i = 0..N):
  xmin:= min(min(xcoords1), min(xcoords2), min(xcoords3)):
  xmax:= max(max(xcoords1), max(xcoords2), max(xcoords3)):
  ycoords1:= seq(simplexlist[i][1][2], i = 0..N):
  ycoords2:= seq(simplexlist[i][2][2], i = 0..N):
  ycoords3:= seq(simplexlist[i][3][2], i = 0..N):
  ymin:= min(min(ycoords1), min(ycoords2), min(ycoords3)):
  ymax:= max(max(ycoords1), max(ycoords2), max(ycoords3)):
> xplotmin:=xmin-(xmax-xmin)/10.0:
  xplotmax:=xmax+(xmax-xmin)/10.0:
  yplotmin:=ymin-(ymax-ymin)/10.0:
  yplotmax:=ymax+(ymax-ymin)/10.0:
> x:='x': y:='y': #unassign x and y
> fcontour:=contourplot(f(x,y),x=xplotmin..xplotmax,y=yplotmin..
  yplotmax, contours=100, scaling=constrained, numpoints=5000):
> radius:= min(xmax-xmin,ymax-ymin)/200.0:
> bestpointplot:= circle(lastpoint, radius, color=blue,thickness=
  10):
> display(seq(triangles[i],i=0..N),fcontour,bestpointplot);
```




Final Answer

The minimum value,

```
> ff(lastpoint);
```

$$0.2054289128 \quad (10.1)$$

occurs at

```
> lastpoint;
```

$$[1.208509662, 1.208141869] \quad (10.2)$$

**Answer: Converges to [1.208509662, 1.208141869]**
**about (**epsilon := 1.0 * 10^(-5)) **5 correct digital places**
##############################################################
**7. Apply the Nelder–Mead Method to find the minimum of the Rosenbrock function f (x,y) =** $\mathbf{100 \cdot \left(y - x^2\right)^2 + (x-1)^2}$

$$[1.208509662, 1.208141869]$$

$$100\left(-x^2+y\right)^2+(x-1)^2 \quad (11)$$

```
> f := (x, y) -> evalf(abs(100·(y - x^2)^2 + (x - 1)^2))
```

$$f := (x, y) \mapsto evalf\left(\left|100\,\left(y - x^2\right)^2 + (x-1)^2\right|\right) \quad \textbf{(12)}$$

```
> xmin:= 0.5; xmax:= 1.5; ymin:= 0.5; ymax:= 1.5;
```

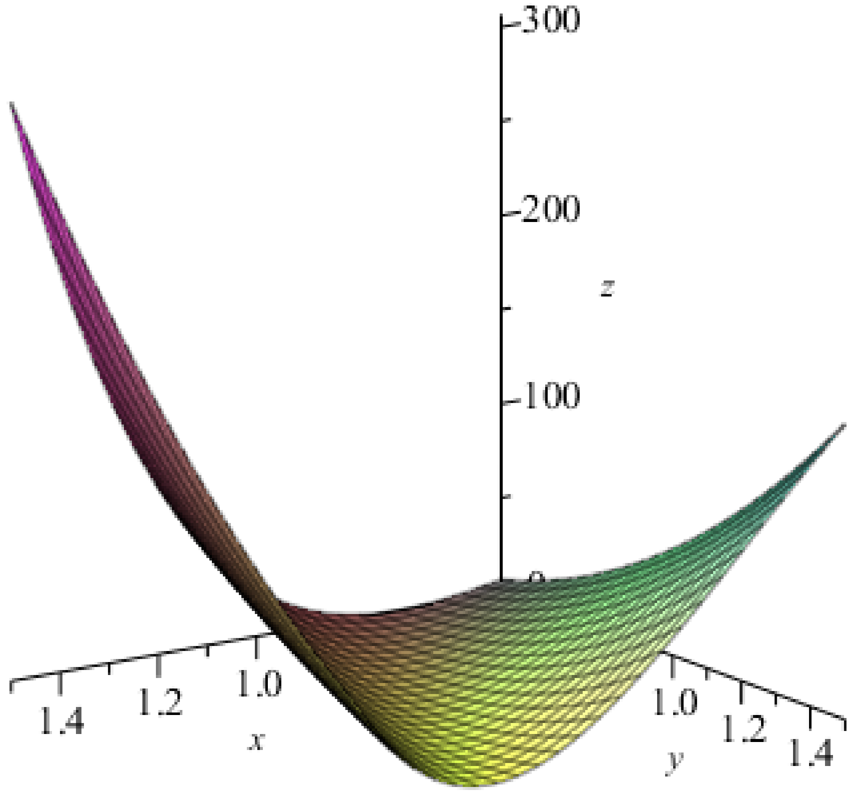
$$xmin := 0.5$$

$$xmax := 1.5$$

$$ymin := 0.5$$

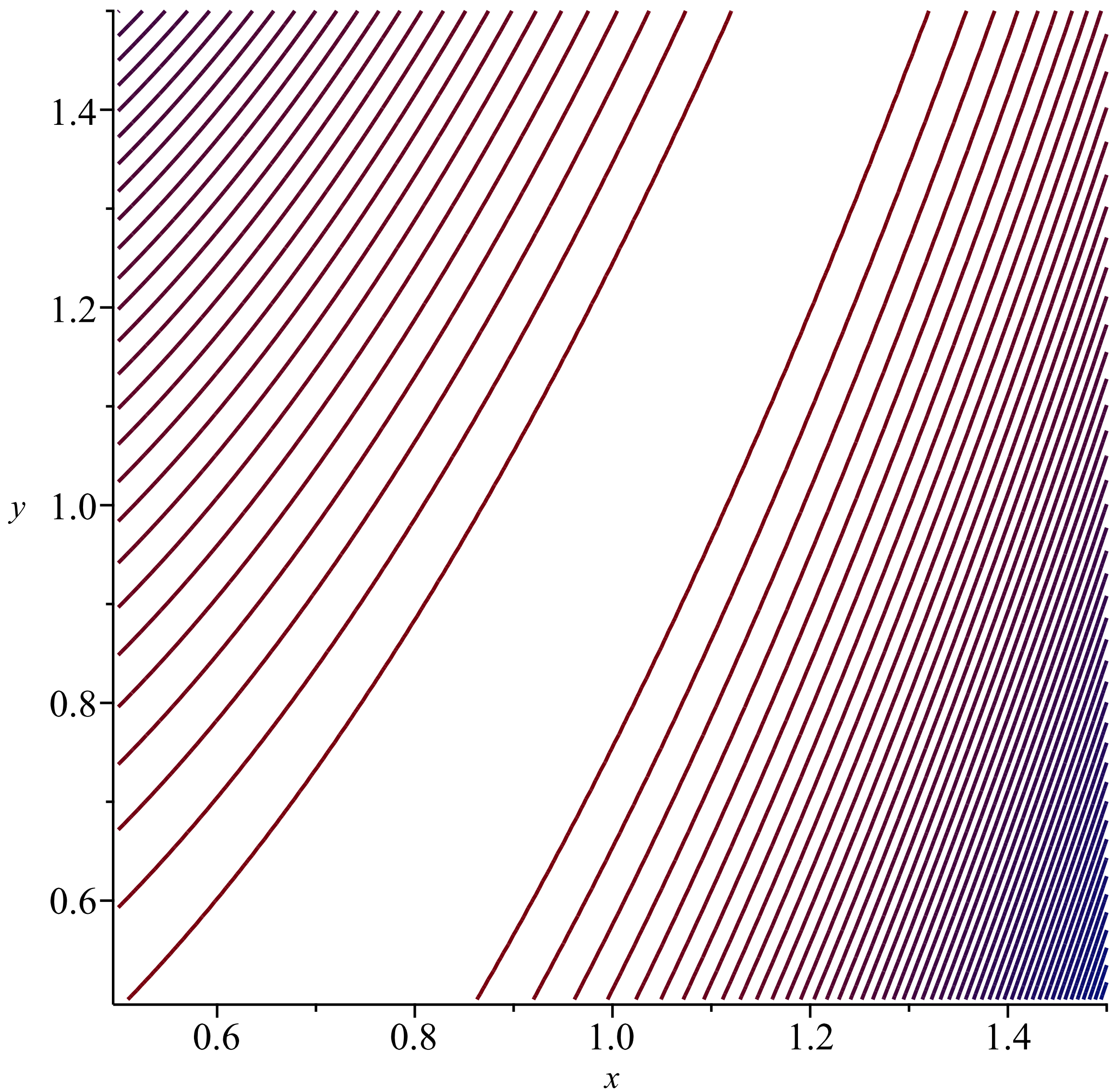
$$ymax := 1.5 \quad \textbf{(13)}$$

```
> with(plots):
> plot3d(f(x,y),x=xmin..xmax,y=ymin..ymax,axes=normal,labels=[x,y,
  z]);
```




```
>
>
```

```
> contourplot(f(x,y),x=xmin..xmax,y=ymin..ymax,scaling=
  constrained,contours=50, numpoints=5000);
```




Enter initial simplex, stopping tolerance

```
> y1:= [1.5, 0]; y2:= [2, 0]; y3:=[2,0.5]; #define initial
  simplex.  Need not be ordered
```

$$y1 := [1.5, 0]$$

$$y2 := [2, 0]$$

$$y3 := [2, 0.5] \tag{12.1}$$

```
> epsilon := 1.0 * 10^(-5); #stopping tolerance (see method
  summary: Termination)
```

$$\epsilon := 0.00001000000000 \tag{12.2}$$

Create function acting on points

```
> ff:=proc(y1)
     f(y1[1],y1[2]);
  end:
```

Procedure for ordering points by objective function values

```
> order3:=proc(g,y1,y2,y3)  #returns list of the 3 points in
  order of increasing g
  local x1, x2, x3,f1, f2, f3:
    f1:=g(y1);f2:=g(y2);f3:=g(y3);
    if (f1 < f2) then
      if (f2 < f3) then
        RETURN([y1, y2,y3]);
      else
        if (f1 < f3) then  #order is f1, f3, f2
          RETURN([y1, y3, y2]);
        else #order is is f3, f1, f2
          RETURN([y3, y1, y2]);
        end if:
     end if:
  else
     if (f1 < f3) then #order is f2, f1, f3
       RETURN([y2, y1, y3]);
     else
       if (f3 < f2) then #order is f3, f2, f1
         RETURN([y3, y2, y1]);
       else #order is f2, f3, f1
         RETURN([y2, y3, y1]);
       end if:
     end if:
  end if:
  end: #end procedure
```

Create first simplex

```
> xlist:=order3(ff,y1, y2, y3):
  x1:=xlist[1]: x2:=xlist[2]: x3:=xlist[3]:
```

Procedure for deciding which action to perform on simplex

```
> nm:=proc(ff,y1,y2,y3,ep)   #takes func., ordered simplex, tol.,
  outputs lambda only (or 'shrink', or 'STOP')
    local x, deltax, fy1, fy2, fy3, fminus, fzero, fplus, fone,
  ftwo:
    ######
    x := (y1+y2)/2.0:
    deltax := x - y3:
    fy1 := ff(y1):
    fy2 := ff(y2):
    fy3 := ff(y3):
    fminus := ff(x + (-1/2)*deltax):
    fzero := ff(x):
    fplus := ff(x + (1/2)*deltax):
    fone := ff(x + deltax):
    ftwo := ff(x + 2*deltax):
    ######
    if (max(abs(fzero - fy1), abs(fzero - fy3)) < ep) then
      return 'STOP':
    elif (fone < fy1) then
      if (ftwo < fone) then
        return 2;
      else
```

```
        return 1;
      end if:
    elif (fone < fy2) then
      return 1;
    elif (fone < fy3) then #try contraction w/ lambda = +1/2
      if (fplus < fy3) then
        return 0.5:
      else
        return 'shrink':
      end if:
    else #try contraction w/lambda = -1/2
      if (fminus < fy3) then
        return -0.5:
      else
        return 'shrink':
      end if:
    end if:
  end:   #end procedure
```

Execution of algorithm

```
> simplexlist[0]:=[x1,x2,x3]:
> lambda:=0:
> N:=0: #  # of iterations
> for i from 0 while ((i < 50) and not(lambda = 'STOP')) do
    N:=i:   #note the i<50 above.  Otherwise roundoff error can
  cause infinite loop.
    lambda:=nm(ff, x1, x2, x3, epsilon):
    lambdalist[i]:=lambda:
    if (lambda = 'STOP') then
      actionlist[i]:="STOP":
      x := (x1 + x2)/2.0:
      if (ff(x) < ff(x1)) then
        lastpoint:=x:
        minf := ff(x):
      else
        lastpoint:= x1:
        minf := ff(x1):
      end if:
    elif (lambda = 'shrink') then
      actionlist[i]="Shrink":
      x2:=(x1+x2)/2.0:
      x3:=(x1+x3)/2.0:
      newsimplex := order3(ff, x1, x2, x3):
      x1:=newsimplex[1]:
      x2:=newsimplex[2]:
      x3:=newsimplex[3]:
      simplexlist[i+1]:=[x1,x2,x3]:
    else
      if (lambda = 1) then
        actionlist[i]:="Reflect":
      elif (lambda = 1/2) then
        actionlist[i]:="Outside Contraction":
      else
        actionlist[i]:="Inside Contraction":
      end if:
      x := (x1 + x2)/2.0:
```

```
      deltax := x - x3:
      x3 := x + lambda*deltax:
      newsimplex := order3(ff, x1, x2, x3):
      x1:=newsimplex[1]:
      x2:=newsimplex[2]:
      x3:=newsimplex[3]:
      simplexlist[i+1]:=[x1,x2,x3]:
    end if:
  end do:
```

Print list of simplices, actions

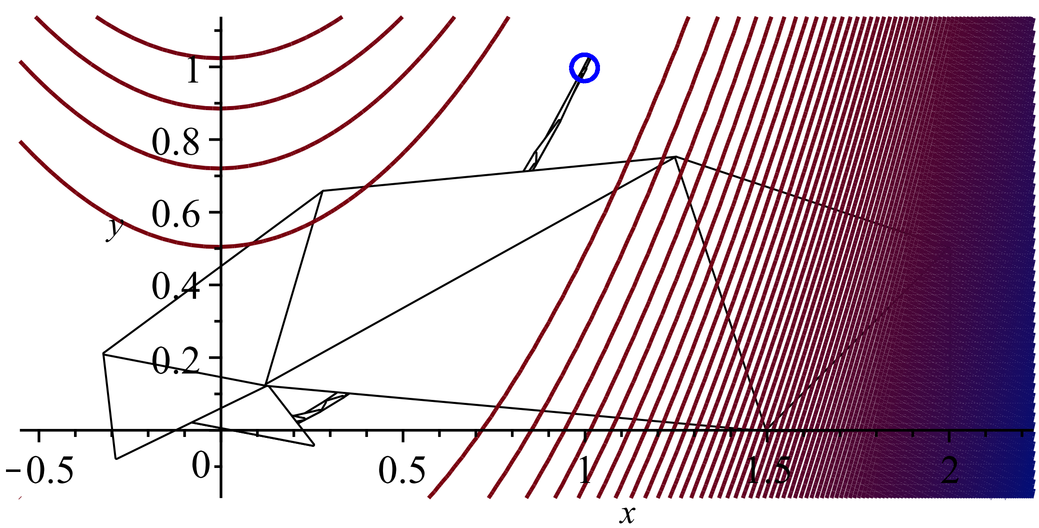
```
> for i from 0 while (i < N) do
  printf("Simplex Number %d: (%f,%f),(%f,%f),(%f,%f) %s\n", i,
  simplexlist[i][1][1],simplexlist[i][1][2],
  simplexlist[i][2][1],simplexlist[i][2][2],
  simplexlist[i][3][1],simplexlist[i][3][2],
  actionlist[i]);
  end do:
Simplex Number 0: (1.500000,0.000000),(2.000000,0.500000),
(2.000000,0.000000) Inside Contraction
Simplex Number 1: (1.250000,0.750000),(1.500000,0.000000),
(2.000000,0.500000) Inside Contraction
Simplex Number 2: (0.125000,0.125000),(1.250000,0.750000),
(1.500000,0.000000) Outside Contraction
Simplex Number 3: (0.125000,0.125000),(0.281250,0.656250),
(1.250000,0.750000) Outside Contraction
Simplex Number 4: (0.125000,0.125000),(-0.320312,0.210938),
(0.281250,0.656250) Outside Contraction
Simplex Number 5: (0.125000,0.125000),(-0.320312,0.210938),
(-0.287109,-0.076172) Inside Contraction
Simplex Number 6: (-0.192383,0.045898),(0.125000,0.125000),
(-0.320312,0.210938) Reflect
Simplex Number 7: (-0.192383,0.045898),(0.252930,-0.040039),
(0.125000,0.125000) Inside Contraction
Simplex Number 8: (0.077637,0.063965),(-0.192383,0.045898),
(0.252930,-0.040039) Inside Contraction
Simplex Number 9: (0.097778,0.007446),(0.077637,0.063965),
(-0.192383,0.045898) Inside Contraction
Simplex Number 10: (0.097778,0.007446),(0.077637,0.063965),
(-0.052338,0.040802) Reflect
Simplex Number 11: (0.227753,0.030609),(0.097778,0.007446),
(0.077637,0.063965) Inside Contraction
Simplex Number 12: (0.227753,0.030609),(0.097778,0.007446),
(0.120201,0.041496) Outside Contraction
Simplex Number 13: (0.227753,0.030609),(0.184048,0.007793),
(0.097778,0.007446) Inside Contraction
Simplex Number 14: (0.227753,0.030609),(0.151839,0.013324),
(0.184048,0.007793) Reflect
Simplex Number 15: (0.227753,0.030609),(0.195544,0.036139),
(0.151839,0.013324) Reflect
Simplex Number 16: (0.271458,0.053425),(0.227753,0.030609),
(0.195544,0.036139) Inside Contraction
Simplex Number 17: (0.271458,0.053425),(0.222575,0.039078),
(0.227753,0.030609) Inside Contraction
Simplex Number 18: (0.285543,0.077536),(0.271458,0.053425),
(0.222575,0.039078) Reflect
```

```
Simplex Number 19: (0.334426,0.091883),(0.285543,0.077536),
(0.271458,0.053425) Inside Contraction
Simplex Number 20: (0.387039,0.147279),(0.334426,0.091883),
(0.285543,0.077536) Reflect
Simplex Number 21: (0.387039,0.147279),(0.435922,0.161626),
(0.334426,0.091883) Reflect
Simplex Number 22: (0.488534,0.217022),(0.387039,0.147279),
(0.435922,0.161626) Reflect
Simplex Number 23: (0.488534,0.217022),(0.439651,0.202676),
(0.387039,0.147279) Reflect
Simplex Number 24: (0.541147,0.272418),(0.488534,0.217022),
(0.439651,0.202676) Inside Contraction
Simplex Number 25: (0.541147,0.272418),(0.477246,0.223698),
(0.488534,0.217022) Inside Contraction
Simplex Number 26: (0.550521,0.310130),(0.541147,0.272418),
(0.477246,0.223698) Reflect
Simplex Number 27: (0.614421,0.358851),(0.550521,0.310130),
(0.541147,0.272418) Inside Contraction
Simplex Number 28: (0.665119,0.458635),(0.614421,0.358851),
(0.550521,0.310130) Reflect
Simplex Number 29: (0.729020,0.507355),(0.665119,0.458635),
(0.614421,0.358851) Inside Contraction
Simplex Number 30: (0.862367,0.731283),(0.729020,0.507355),
(0.665119,0.458635) Inside Contraction
Simplex Number 31: (0.862367,0.731283),(0.730406,0.538977),
(0.729020,0.507355) Reflect
Simplex Number 32: (0.862367,0.731283),(0.863753,0.762905),
(0.730406,0.538977) Inside Contraction
Simplex Number 33: (0.862367,0.731283),(0.863753,0.762905),
(0.796733,0.643036) Reflect
Simplex Number 34: (0.929386,0.851153),(0.862367,0.731283),
(0.863753,0.762905) Inside Contraction
Simplex Number 35: (0.879815,0.777062),(0.929386,0.851153),
(0.862367,0.731283) Inside Contraction
Simplex Number 36: (0.989068,0.979755),(0.879815,0.777062),
(0.929386,0.851153) Inside Contraction
Simplex Number 37: (0.989068,0.979755),(0.931914,0.864781),
(0.879815,0.777062) Inside Contraction
Simplex Number 38: (0.989068,0.979755),(0.931914,0.864781),
(0.920153,0.849665) Reflect
Simplex Number 39: (0.989068,0.979755),(1.000829,0.994871),
(0.931914,0.864781) Inside Contraction
Simplex Number 40: (0.989068,0.979755),(0.963431,0.926047),
(1.000829,0.994871) Inside Contraction
Simplex Number 41: (0.989068,0.979755),(0.988539,0.973886),
(0.963431,0.926047) Reflect
Simplex Number 42: (1.014176,1.027594),(0.989068,0.979755),
(0.988539,0.973886) Inside Contraction
Simplex Number 43: (0.995081,0.988780),(1.014176,1.027594),
(0.989068,0.979755) Inside Contraction
Simplex Number 44: (0.996848,0.993971),(0.995081,0.988780),
(1.014176,1.027594) Inside Contraction
Simplex Number 45: (0.996848,0.993971),(1.005070,1.009485),
(0.995081,0.988780) Outside Contraction
Simplex Number 46: (0.996848,0.993971),(1.003899,1.008202),
(1.005070,1.009485) Inside Contraction
Simplex Number 47: (1.002722,1.005286),(0.996848,0.993971),
```

```
(1.003899,1.008202) Outside Contraction
Simplex Number 48: (0.997728,0.995342),(1.002722,1.005286),
(0.996848,0.993971) Inside Contraction
```

Draw simplices

```
> with(plottools):
> for i from 0 while (i < N+1) do
    triangles[i] := polygon([simplexlist[i][1], simplexlist[i]
  [2],  simplexlist[i][3]], color=white,labels=[x,y]):
  end do:
> xcoords1:= seq(simplexlist[i][1][1], i = 0..N):  #define
  plotting window
  xcoords2:= seq(simplexlist[i][2][1], i = 0..N):
  xcoords3:= seq(simplexlist[i][3][1], i = 0..N):
  xmin:= min(min(xcoords1), min(xcoords2), min(xcoords3)):
  xmax:= max(max(xcoords1), max(xcoords2), max(xcoords3)):
  ycoords1:= seq(simplexlist[i][1][2], i = 0..N):
  ycoords2:= seq(simplexlist[i][2][2], i = 0..N):
  ycoords3:= seq(simplexlist[i][3][2], i = 0..N):
  ymin:= min(min(ycoords1), min(ycoords2), min(ycoords3)):
  ymax:= max(max(ycoords1), max(ycoords2), max(ycoords3)):
> xplotmin:=xmin-(xmax-xmin)/10.0:
  xplotmax:=xmax+(xmax-xmin)/10.0:
  yplotmin:=ymin-(ymax-ymin)/10.0:
  yplotmax:=ymax+(ymax-ymin)/10.0:
> x:='x': y:='y': #unassign x and y
> fcontour:=contourplot(f(x,y),x=xplotmin..xplotmax,y=yplotmin..
  yplotmax, contours=100, scaling=constrained, numpoints=5000):
> radius:= min(xmax-xmin,ymax-ymin)/200.0:
> bestpointplot:= circle(lastpoint, radius, color=blue,thickness=
  10):
> display(seq(triangles[i],i=0..N),fcontour,bestpointplot);
```




Final Answer

The minimum value,

```
> ff(lastpoint);
```

$$2.585664334\ 10^{-6} \tag{20.1}$$

occurs at

```
> lastpoint;
```

$$[0.9985368192, 0.9971424701] \tag{20.2}$$

```
>
```

**Answer: Converges to [1.0 , 1.0]**

**############################################################**

## Section 13.2

**############################################################**

**1. Use Newton's Method to find the minimum of**

**f(x,y) =** $\exp(-x^2y^2) + (x-1)^2 + (y-1)^2$

**various initial conditions, and compare answers. How many correct digits can you obtain with this method?**

```
> restart;
> with(plots, implicitplot) : with(plots) : with(LinearAlgebra) : with(VectorCalculus) :
> newtonMD :=proc(X0, Y0, TOL, N)
      local i, s, x0, y0, sol, A;
      x0 := X0; y0 := Y0; sol := <x0, y0>;
      i := 1;
      while i ≤ N do
         A := Hessian(F(x, y), [x, y] = [x0, y0]);
         if Determinant(A) = 0 then
              printf("Non-invertible Jacobian. Method failed");
              break;
          else
          s := LinearSolve(A, subs([x = x0, y = y0], <-diff(F(x, y), x), -diff(F(x, y), y)>));
          sol := sol + s;
          x0 := DotProduct(sol, <1, 0>); y0 := DotProduct(sol, <0, 1>);
          printf("Step %d: x=%.8g, y=%.8g \n", i, x0, y0);
          if evalf(Norm(s)) < TOL then
             printf("Number of iterations needed: %d", i);return(F(x0, y0));
             break;
           end if;
         i := i + 1;
         end if;
     end do;
     printf("The method failed after %d iterations.\n", N);
     return( );
  end proc:
> F := (x, y) → exp(-x^2·y^2) + (x − 1)^2 + (y − 1)^2;
```

$$F := (x, y) \mapsto e^{-x^2 y^2} + (x-1)^2 + (y-1)^2 \qquad \textbf{(14)}$$

```
> f := newtonMD(0.0, 0.0, 0.5·10^-6, 100)
Step 1: x=1, y=1
Step 2: x=1.2689414, y=1.2689414
Step 3: x=1.2074557, y=1.2074557
Step 4: x=1.2088175, y=1.2088175
Step 5: x=1.2088176, y=1.2088176
Number of iterations needed: 5
```

$$f := 0.205427886650958 \qquad \textbf{(15)}$$

```
> f := newtonMD(1.0, 1.0, 0.5·10^-6, 100)
Step 1: x=1.2689414, y=1.2689414
Step 2: x=1.2074557, y=1.2074557
Step 3: x=1.2088175, y=1.2088175
Step 4: x=1.2088176, y=1.2088176
Number of iterations needed: 4
```

$$f := 0.205427886650958 \qquad \textbf{(16)}$$

```
> f := newtonMD(-1.5, -0.5, 0.5·10^-6, 100)
Step 1: x=1.3565272, y=1.1609391
Step 2: x=1.20287, y=1.2093303
Step 3: x=1.2088076, y=1.2088217
Step 4: x=1.2088176, y=1.2088176
Step 5: x=1.2088176, y=1.2088176
```

```
Number of iterations needed: 5
```
$$f := 0.205427886650958 \tag{17}$$

> $f := newtonMD(1.5, 1.5, 0.5 \cdot 10^{-6}, 100)$

```
Step 1: x=1.1157599, y=1.1157599
Step 2: x=1.2128122, y=1.2128122
Step 3: x=1.208816, y=1.208816
Step 4: x=1.2088176, y=1.2088176
Step 5: x=1.2088176, y=1.2088176
Number of iterations needed: 5
```
$$f := 0.205427886650958 \tag{18}$$

> $f := newtonMD(2.0, 2.0, 0.5 \cdot 10^{-6}, 100)$

```
Step 1: x=1.0000284, y=1.0000284
Step 2: x=1.2689127, y=1.2689127
Step 3: x=1.2074575, y=1.2074575
Step 4: x=1.2088175, y=1.2088175
Step 5: x=1.2088176, y=1.2088176
Number of iterations needed: 5
```
$$f := 0.205427886650958 \tag{19}$$

> $\boldsymbol{Minimize(F(x, y), x = 0 ..2, y = 0 ..2)}$

$$[0.205427886650958014, [x = 1.20881758948871, y = 1.20881758948871]] \tag{20}$$

>

**Answer:**
**Different Initial conditions give the same results.**
**Neton Method and Minimize function give the same result**
$\boldsymbol{[0.2054279, [x = 1.2088176, y = 1.208818]]}$
**Newton's Method will be accurate to machine precision,**
**since it is finding a simple root. Steepest Descentwill have**
**error of size ≈ epsilon^(1/2 )**
##################################################################

**2.** ***Apply Newton's Method*** **to** ***find the minima of the following functions*** **to** ***six correct decimal places·*** **(*each function has two minima*) :**

**(*a*)** $\boldsymbol{f(x, y) = x^4 + y^4 + 2x^2y^2 + 6xy - 4x - 4y + 1}$

> $with(Optimization);$

$$[ImportMPS, Interactive, LPSolve, LSSolve, Maximize, Minimize, NLPSolve, QPSolve] \tag{21}$$

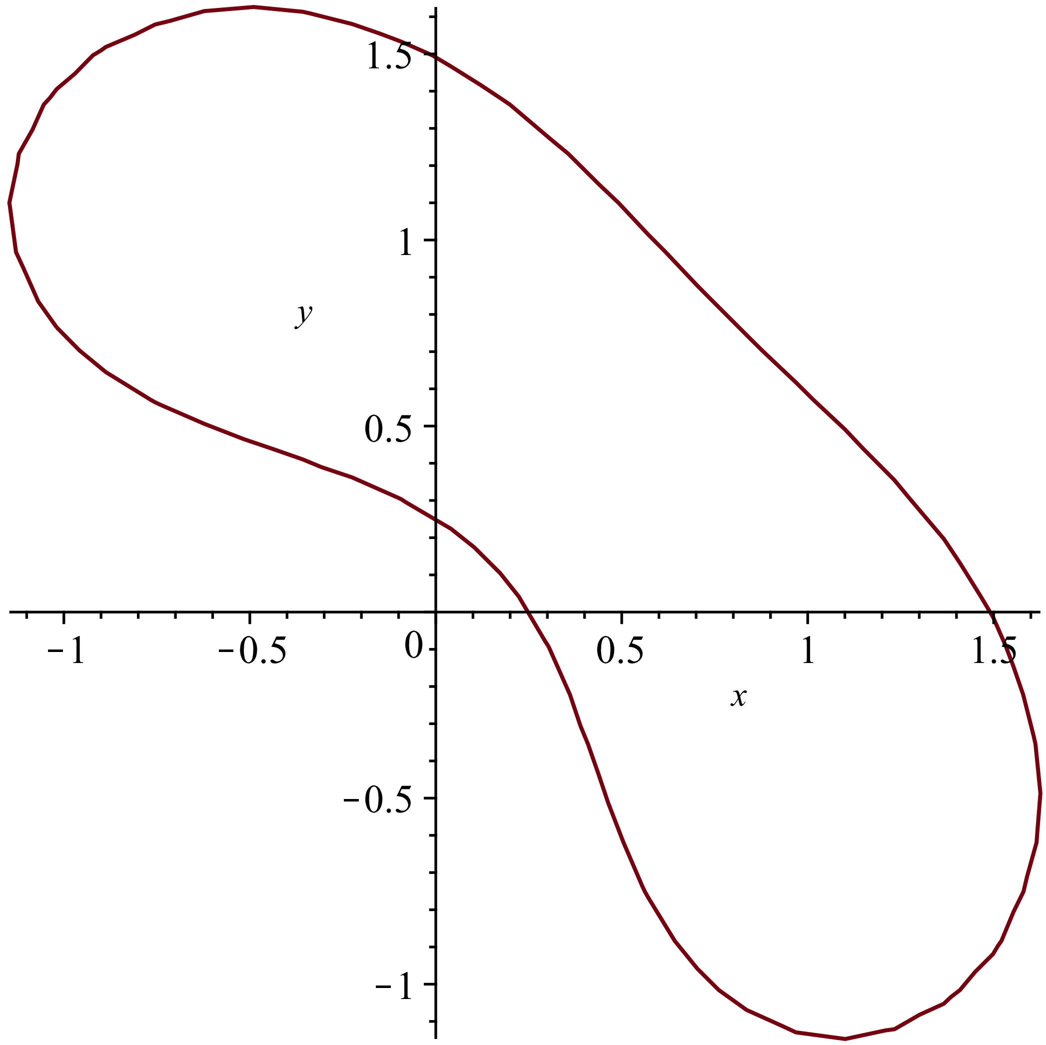
>

>

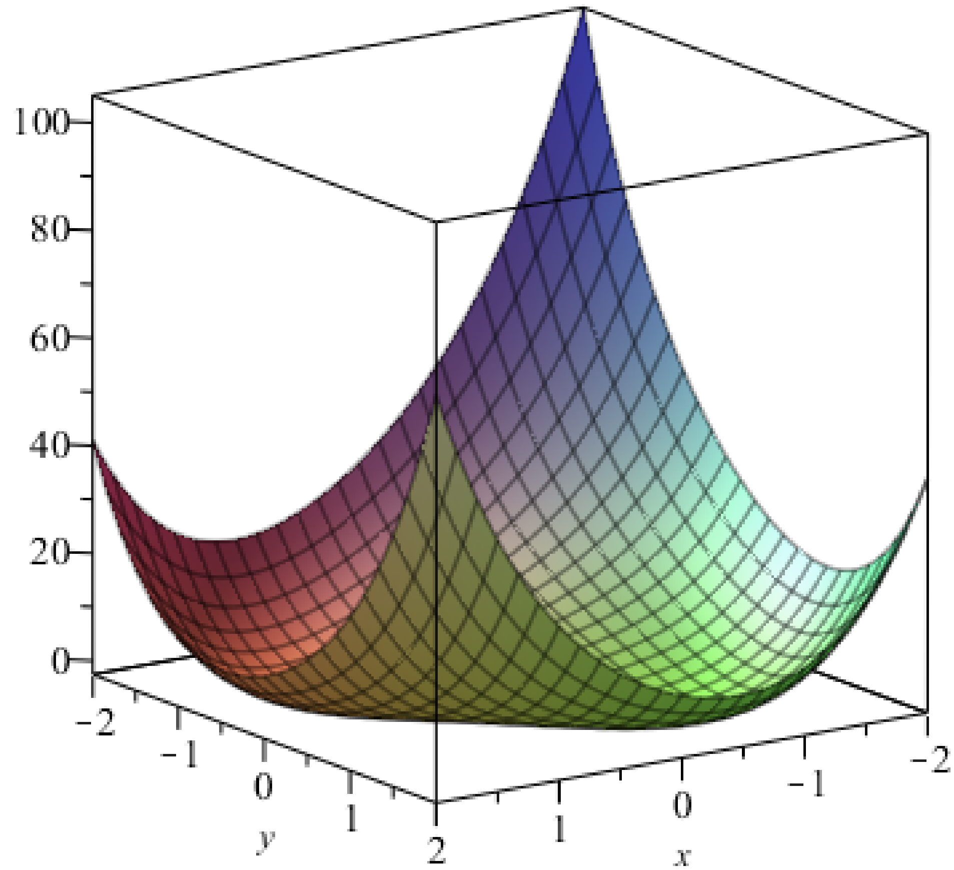
> $F := (x, y) \rightarrow x^4 + y^4 + 2x^2 \cdot y^2 + 6x \cdot y - 4x - 4y + 1$

$$F := (x, y) \mapsto x^4 + y^4 + 2x^2y^2 + 6xy + (-4x) + (-4y) + 1 \tag{22}$$

> $implicitplot(x^4 + y^4 + 2x^2 \cdot y^2 + 6x \cdot y - 4x - 4y + 1 = 0, x = -1.5 ..2.0, y = -1.5 ..2.5, scaling = constrained);$




> $plot3d\left(x^4 + y^4 + 2\,x^2\cdot y^2 + 6\,x\cdot y - 4\,x - 4\,y + 1,\, x = -2\,..2,\, y = -2\,..2,\, axes = boxed\right)$




> $f := newtonMD(0.5, 1.5, 0.5 \cdot 10^{-6}, 100);$

```
Step 1: x=-0.083333333, y=1.25
Step 2: x=-0.50189628, y=1.1869123
Step 3: x=-0.47009898, y=1.1359929
Step 4: x=-0.46600761, y=1.1326596
Step 5: x=-0.46597192, y=1.1326386
Step 6: x=-0.46597192, y=1.1326386
Number of iterations needed: 6
```

$$f := -2.58333333333333 \tag{23}$$

> $f := newtonMD(1.5, 0.5, 0.5 \cdot 10^{-6}, 100);$

```
Step 1: x=1.25, y=-0.083333333
Step 2: x=1.1869123, y=-0.50189628
Step 3: x=1.1359929, y=-0.47009898
Step 4: x=1.1326596, y=-0.46600761
Step 5: x=1.1326386, y=-0.46597192
Step 6: x=1.1326386, y=-0.46597192
Number of iterations needed: 6
```

$$f := -2.58333333333333 \tag{24}$$

> $Minimize(F(x, y), x = -2 ..0, y = 0 ..2)$

$$[-2.58333333333333393, [x = -0.465971920673358, y = 1.13263858739415]] \tag{25}$$

> $Minimize(F(x, y), x = 0 ..2, y = -2 ..0)$

$$[-2.58333333333333393, [x = 1.13263858739415, y = -0.465971920673358]] \tag{26}$$

**Answer:**

**Minima is :**

$$[-2.58333333333333393, [x = -0.465971920673358, y = 1.13263858739415]]$$
$$[-2.58333333333333393, [x = 1.13263858739415, y = -0.465971920673358]]$$

**Both Neton Method and Minimize gives the same results**

**##########################################################################**

>