HW2

March 24, 2024

1 Homework Assignment 2

[4]:
```
from sympy import*
import numpy as np
from sympy.abc import x,y,z,u,v,w,t
from dtumathtools import *
from IPython.display import display, Latex, Image
```

1.1 Problem 1

A function $f : \mathbb{R} \to \mathbb{R}$ is given by the expression:

$$f(x) = \cos(3x).$$

a) Determine the approximating polynomial P_{4,f,x_0} of degree (at most) 4 with expansion point $x_0 = 0$.

b) Use the polynomial you found to compute an approximate value of $\cos(\frac{1}{2})$, and estimate how far the found value is from the exact value.

We are given the following information about a function g:

$$g(0) = 1 \quad \text{and} \quad g^{(n)}(0) = \begin{cases} (-1)^{\frac{n}{2}} 2^n & n \text{ is even,} \\ 0 & n \text{ is odd.} \end{cases}$$

c) Determine the approximating polynomial P_{7,g,x_0} of g of degree (at most) 7 with expansion point $x_0 = 0$. Give your best suggestion on which function g might be.

[4]:
```
f = cos(3*x)  #define the function
P_4 = series(f,x0=0,n=5).removeO()  #determining the approximation polynomial of␣
↪degree 4
P_4
```

[4]:
$$\frac{27x^4}{8} - \frac{9x^2}{2} + 1$$

To evaluate $\cos(\frac{1}{2})$ we need to find such x that $3x = \frac{1}{2}$, $x = \frac{1}{6}$.

```
[54]: display(Latex(fr'$\cos (\frac{1}{2} ) = $ {P_4.subs(x,1/6)}')) #substitute x = 1/
      ↪6 into P_4 expression and display it
```

$\cos(\frac{1}{2}) = 0.877604166666667$

The error estimation is then $R_{n,f,x_0} = f_\xi^{(n+1)} \frac{1}{(n+1)!}(x - x_0)^{n+1}$, where $\xi \in [x_0, x], n = 4$.

$$R_{4,f,0}(x) = f_\xi^{(5)} \frac{1}{5!}(x)^5 \tag{1}$$

$$f_\xi^{(5)} = -27 * 9 \sin(3\xi), \xi \in [0, \frac{1}{6}] \tag{2}$$

We need to choose ξ such that $|R_{4,f,0}|$ is the maximum possible value on the $[0, \frac{1}{6}]$ interval, therefore $\xi = \frac{1}{6}$.

$$R_{4,f,0}(\frac{1}{6}) = \frac{-27 * 9}{5!} \sin(3\frac{1}{6})(\frac{1}{6})^5 \tag{3}$$

```
[59]: R_4 = -27*9/(120 * 6**5)*sin(0.5)
      abs(R_4)
```

```
[59]: 0.000124850400678178
```

The greates error $R_{4,f,0}(\frac{1}{6}) = 0.0001248$

```
[62]: display(Latex(fr'$\cos(\frac{1}{2}) \in$ [{P_4.subs(x,1/6)+R_4}, {P_4.subs(x,1/
      ↪6)-R_4}]'))
```

$\cos(\frac{1}{2}) \in [0.877479316265988, 0.877729017067345]$

The true value of $\cos(\frac{1}{2})$ is 0.8775826, which is in the found range above.

We are given the following information about a function g:

$$g(0) = 1 \quad \text{and} \quad g^{(n)}(0) = \begin{cases} (-1)^{\frac{n}{2}} 2^n & n \text{ is even,} \\ 0 & n \text{ is odd.} \end{cases}$$

The approximating polynomial P_{7,g,x_0} of g of degree 7 with expansion point $x_0 = 0$ is then will be

$$P_{7,g,0}(x) = g(0) + g^1(0)x + \frac{1}{2}g^2(0)x^2 + \frac{1}{6}g^3(0)x^3 + \frac{1}{24}g^4(0)x^4 + \frac{1}{120}g^5(0)x^5 + \frac{1}{720}g^6(0)x^6 + \frac{1}{5040}g^7(0)x^7 \tag{4}$$

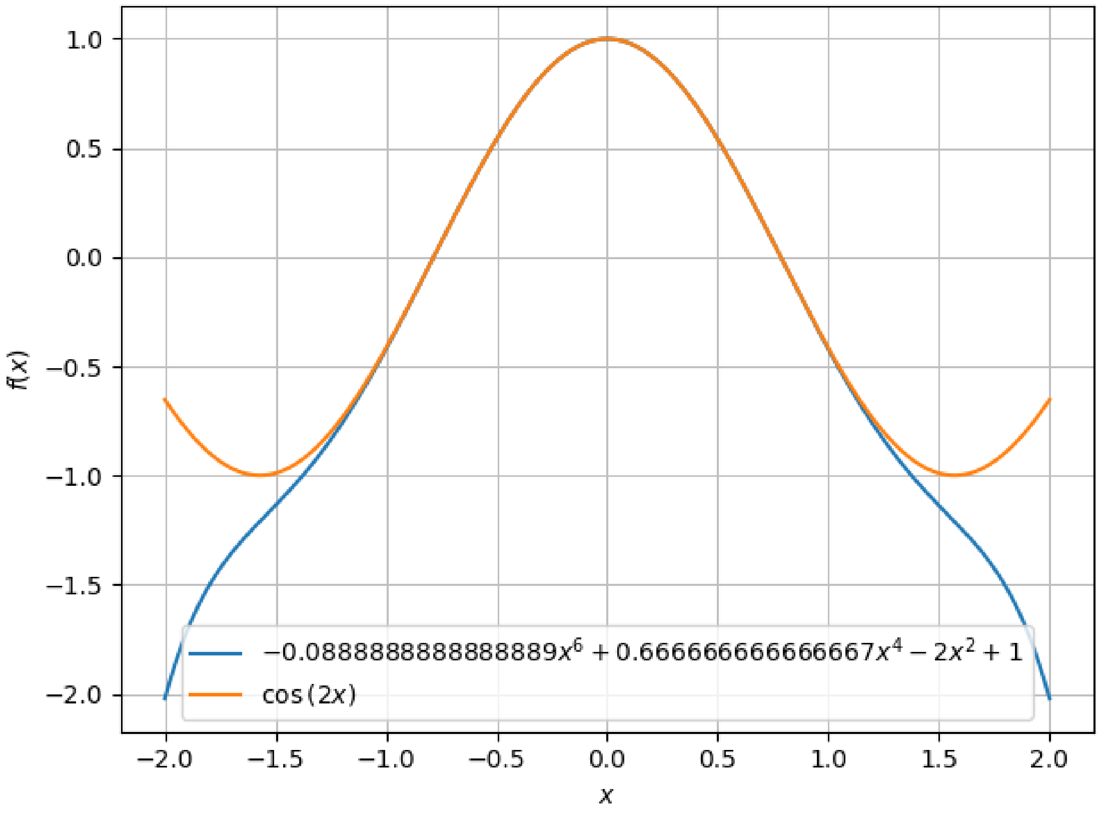
$$P_{7,g,0}(x) = 1 + 0 * x - 2x^2 + \frac{1}{6} * 0 * x^3 + \frac{2}{3}x^4 + \frac{1}{120} * 0 * x^5 - \frac{4}{45}g^6(0)x^6 + \frac{1}{5040} * 0 * x^7 \tag{5}$$

$$P_{7,g,0}(x) = 1 - 2x^2 + \frac{2}{3}x^4 - \frac{4}{45}x^6 \tag{6}$$

The expansion of the function g we got above is very similar to the expansion of the f function, so the g function can also be some sort of cos. If we take a look at the $-2x^2$ term, which equals $g^{(2)}(0)$, and assuming $g(x) = \cos(ax)$, the a should be 2.

We can check it by plotting the expansion and the function on the interval $[x_1, x_2]$ near the point x_0.

[82]:
```
P_7 = 1-2*x**2 + 2/3*x**4 - 4/45*x**6
fig = dtuplot.plot(P_7,(x, -2,2), show=False)
#fig.extend(dtuplot.plot(P_4,(x, -2,2), show=False))
fig.extend(dtuplot.plot(cos(2*x),(x,-2,2),show=False))
fig.show()
```



Based on the graph, our assumption can be considered correct. $g(x) = cos(2x)$

1.2 Problem 2

Let B be the set $B = \{(x_1, x_2) \in \mathbb{R}^2 \mid x_1^2 + x_2^2 \leq 1 \land x_1 \geq 0\}$.

A function $f : B \to \mathbb{R}$ is given by

$$f(x_1, x_2) = x_1^2 x_2^2 + 3x_1x_2 + x_2 - 4.$$

State the image of f.

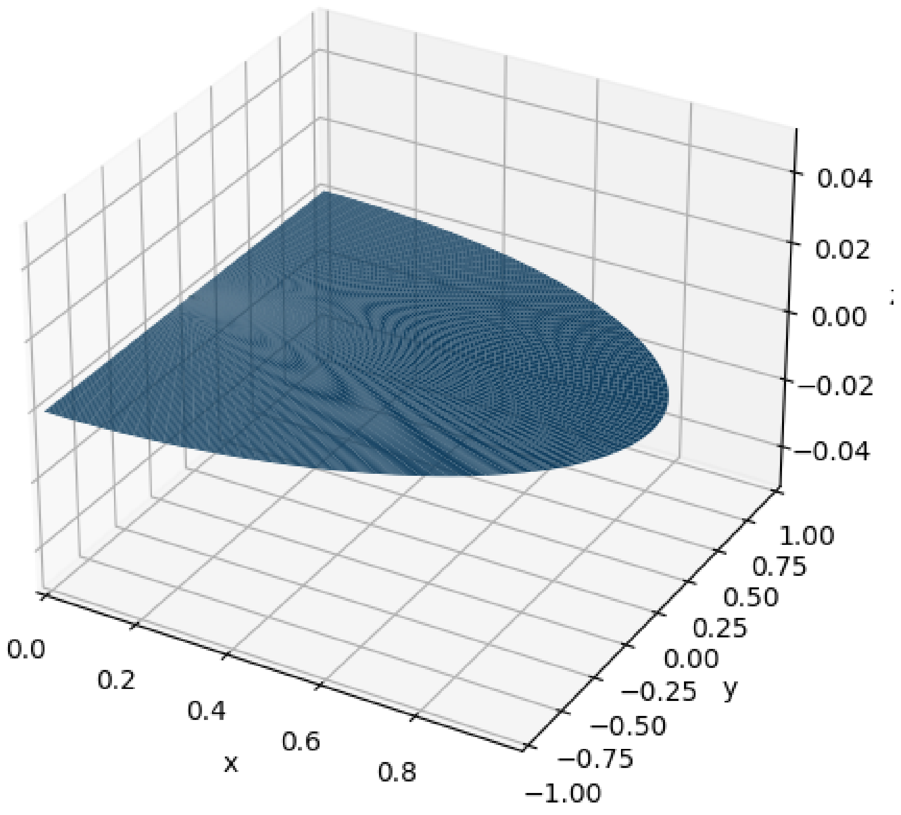
To find the image of f we need to find the local maximum and minimum of the function among the set of points $(x1, x2) \in B$.

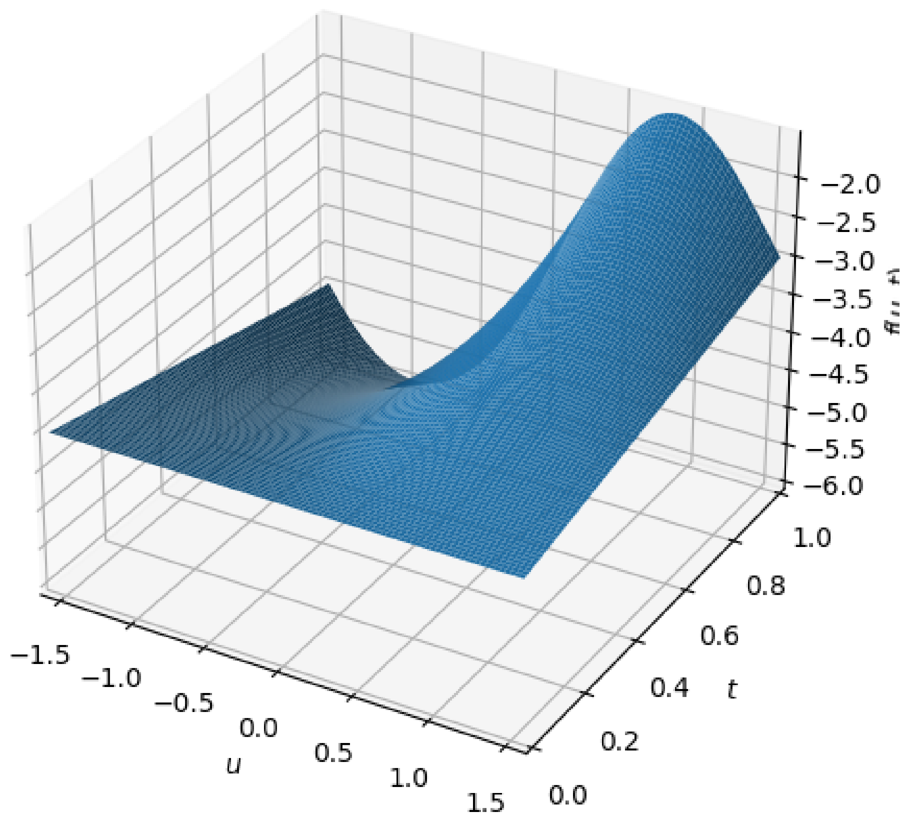
The first step will be defining the parameterization plane. We know that $x_1^2 + x_2^2 \leq 1 \land x_1 \geq 0$ is a half circle.

$$r = [t\ cos(u), t\sin(u)], t \in [0,1], u \in [-\frac{\pi}{2}, \frac{\pi}{2}] \tag{7}$$

To check the parameterization plane we can plot it

```
[6]: r = [t*cos(u), t*sin(u)] # defining a parameterization plane
     f = x**2*y**2 + 3*x*y + y - 4  # defining a function f with x1=x and x2=y
     param = {x: r[0], y: r[1]}   # creating  a dictionary of parametric expressions
     dtuplot.plot3d_parametric_surface(r[0],r[1], 0, (u,-pi/2,pi/2), (t, 0, 1))  #␣
      ↪plotting the plane
     dtuplot.plot3d(f.subs(param), (u,-pi/2,pi/2), (t, 0, 1))  # plotting the␣
      ↪function parameterized by our plane
     #dtuplot.plot3d(f, (x, -5, 5), (y, -5, 5))  # function itself
```





```
[6]: <spb.backends.matplotlib.matplotlib.MatplotlibBackend at 0x76013d7e7f70>
```

From the 2 plots above it is evident that f is continuous and differentiable for all $t \in [0, 1]$ and $u \in [-\frac{\pi}{2}, \frac{\pi}{2}]$.

The next step will be finding stationary points by finding the gradient of the function and equating it to 0.

```
[160]: Nf = dtutools.gradient(f) # finding gradient
       S = Nf.subs(param) # substituting t*cos(u) instead of x and t*sin(u) instead of y
       sol = solve(S, u, t) # solving for t and u
       sol_check = solve(Nf, x, y) # the same thing without substituting␣
        ↪parameterization to check
       sol, sol_check  # displaying solutions
```

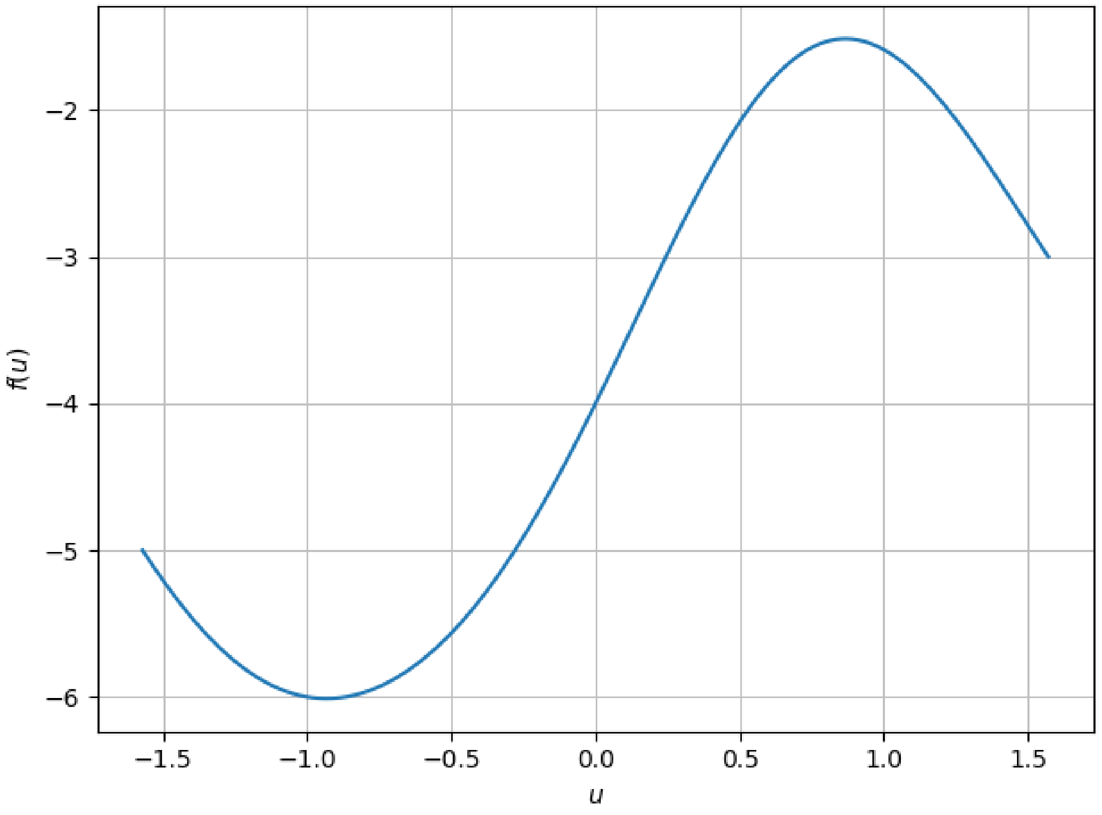
```
[160]: ([(0, -1/3)], [(-1/3, 0)])
```

$u = 0$ and $t = -\frac{1}{3}$, as we can see the t value is out of the range we defined previously, and it doesn't make any sense for the radius to be negative, therefore there are no stationary points.

The third and last step would be boundary investigation, as it is obvious from the plot above that the function has a global maximum and minimum on the boundary.

As it is the boundary of \$ r(u,t) = [t cos(u), t sin(u)], $t \in [0,1], u \in [-\frac{\pi}{2}, \frac{\pi}{2}]$\$, t must be 1. We should have also tested the boundary $r_1(u) = [0, u], u \in [0, 1]$, but as we have plotted the function and it is clear that maximum and minimum values are on the boundary parameterized by the $r(u, 1) = [\cos(u), \sin(u)]$ we won't further investigate the $r_1(u)$ boundary.

```
[10]: param1 = {x: cos(u), y: sin(u)} # creating  a dictionary of parametric␣
       ↪expressions
      dtuplot.plot(f.subs(param1), (u,-pi/2,pi/2)) # plotting the line we got for u in␣
       ↪[-pi/2, pi/2]
      B = dtutools.gradient(f.subs(param1)) # finding gradient(derivative) with␣
       ↪respect to u of the function f
      display(B) #display the derivative
```



$$-2\sin^3(u)\cos(u) - 3\sin^2(u) + 2\sin(u)\cos^3(u) + 3\cos^2(u) + \cos(u)$$

Before we proceed to solve the equation, it is worth mentioning that solving the equation above symbolically, e.g. using `solve()`, yields a system of equations. We would rather use numerical solver `nsolve`.

```
[164]: sol_1 = nsolve(B, u, -1) # find first solution around u=-1
       sol_2 = nsolve(B, u, 0.8) # finding second solution around u=0.8
       sol_1, sol_2
```

```
[164]: (-0.933199076006191, 0.866924817752369)
```

```
[167]: f.subs({x: cos(sol_1), y: sin(sol_1)}), f.subs({x: cos(sol_2), y: sin(sol_2)})
```

```
[167]: (-6.00968504213171, -1.51414118184529)
```

Our image is then $im(f) = [-6.0096, -1.5141]$.

1.3 Problem 3

A function $f : \mathbb{R}^2 \to \mathbb{R}$ is given by

$$f(x_1, x_2) = x_1^2 - 2x_1 + 3x_2^5 - 5x_2^3.$$

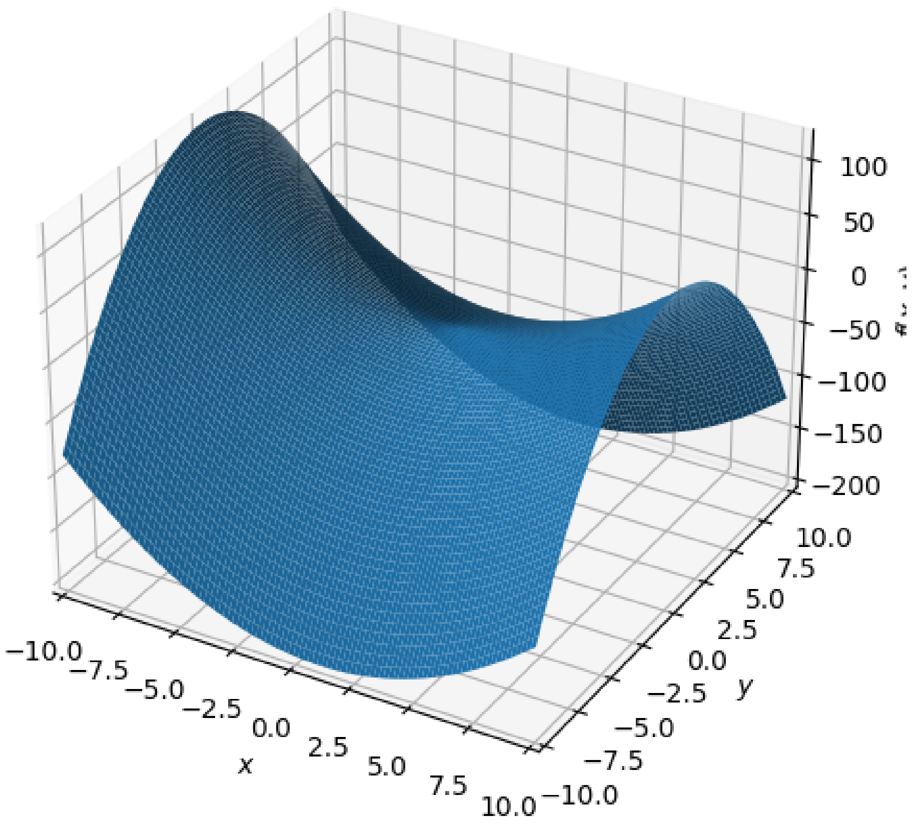
a) Determine all stationary points of f.

b) State whether there is a local maximum, a local minimum, or whether there is a saddle point at the stationary points.

c) Plot for each stationary point the function along with the approximating polynomial P_2 of degree (at most) 2.

As the domain Γ of the function $f : \mathbb{R}^2 \to \mathbb{R}$ is defined on the whole $\mathbb{R}$ we can state that the function doesn't have boundaries. To get a feelig of how the function looks like it's been plotted below.

```
[15]: f = x**2-2*x + 3*y**2 - 5*y**2
      dtuplot.plot3d(f, (x, -10, 10), (y,-10, 10))
```



[15]: <spb.backends.matplotlib.matplotlib.MatplotlibBackend at 0x76013c0f75e0>

Now we can procceed to finding the gradient and solving the system of 2 equations $\begin{cases} \frac{\partial f}{\partial x} = 0 \\ \frac{\partial f}{\partial y} = 0 \end{cases}$

[16]:
```
Nf = dtutools.gradient(f) #finding gradient
extremums = solve(Nf, x,y) #solving the system
display(Nf, extremums)
```

$$\begin{bmatrix} 2x - 2 \\ -4y \end{bmatrix}$$

```
{x: 1, y: 0}
```

It is evident from gradient expression that f is differentiable on the whole Γ. That is the only stationary points exist for this function are the points where the gradient equals to 0. And we have already found such point: $(1, 0)$. The next and the last step will be determination if the point local maximum, minimum or it is a saddle point.

```
[17]: H = dtutools.hessian(f) # finding a hessian matrix of f
      display(H)
```

$$\begin{bmatrix}2 & 0\\0 & -4\end{bmatrix}$$

As the hessian matrix is a diagonal matrix and the diagonal entries have opposite signs, it can be concluded that the function f has one saddle point with coordinates $(1, 0, -1)$.

For convinience we will write a python function that will take a function and expansion point and returns eproximation polynomial of the 2 degree.

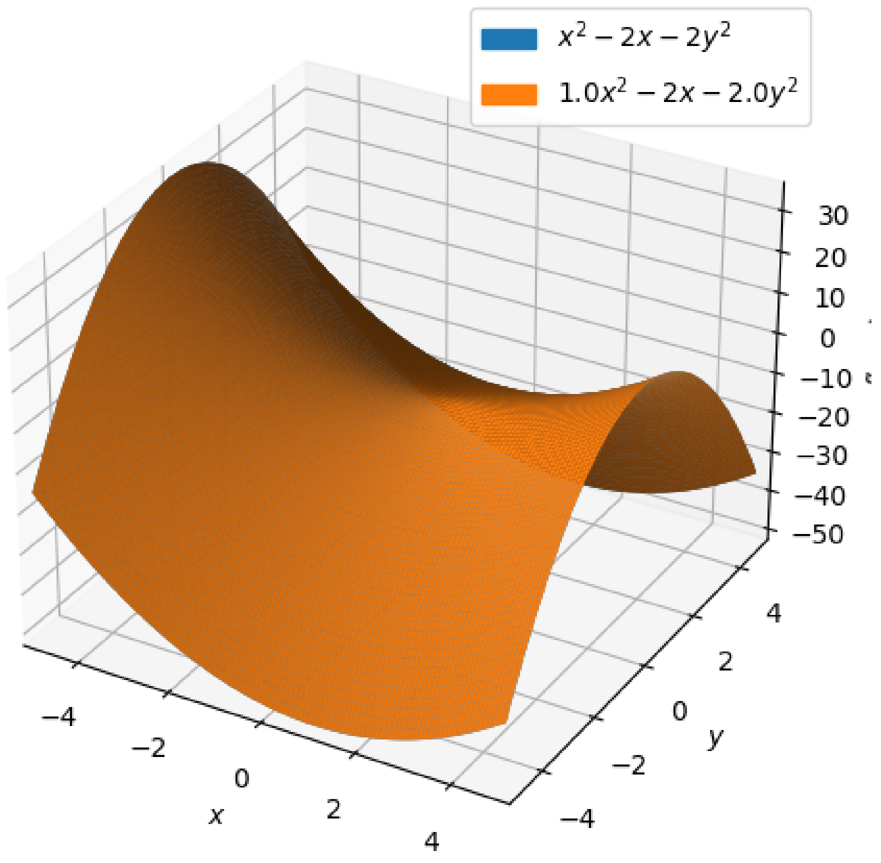
```
[18]: def P(f, x0, y0):
          p = {x: x0, y: y0}
          dx = Matrix([[x-x0],[y-y0]]) #defining x-x0 vector
          Nf = dtutools.gradient(f).subs(p) #finding gradient at the point (x0,y0)
          H = dtutools.hessian(f).subs(p) #finding hessian matrix at the point (x0,y0)
          return f.subs(p) + (Nf.T*dx)[0] + 0.5*dx.dot(H*dx) #returning taylor␣
       ↪approxiamtion

      P2 = P(f, 0, 0)
      display(P2, P2.subs({x: 1 , y: 0}))
```

$1.0x^2 - 2x - 2.0y^2$

-1.0

The approximation is obviously the same because the initial function is a polynomial itself.

```
[20]: xlim = (x, -5, 5)
      ylim = (y,-5, 5)
      pl = dtuplot.plot3d(f, xlim, ylim, show=False) #plot the function
      pl.extend(dtuplot.plot3d(P2, xlim, ylim, show=False)) #plot the approximating␣
       ↪polynomial
      #pl.extend(dtuplot.scatter(np.array([1,0,-1]), rendering_kw={"color":
       ↪'b'},show=False)) #the point
      pl.show()
```



1.4 Problem 4

A function $f : [0, 5] \to \mathbb{R}$ is given by the expression $f(x) = 2x + 3$.

a) Determine, using Python, a value of the Riemann sum over the interval $[0, 5]$ with 30 subintervals, where firstly the left interval end-points of the subintervals are used. Repeat with the right interval end-points.

b) Determine exact values of the Riemann sum over the interval $[0, 5]$ with n subintervals, where firstly the left interval end-points of the subintervals are used. Repeat with the right interval end-points.

c) Provide an exact expression for the maximal and minimal errors. Both expressions may only depend on n.

d) Argue in this special case for the fact that the Riemann sum has the same limit value regardless of the choice of point in the subintervals.

[5]:
```
f = 2*x + 3 #defiing the function
```

The Riemann Sum on the interval $x \in [a, b]$ can be defined as $\sum_{i=1}^{n} f(x_i) \triangle x$, where $\triangle x = \frac{b-a}{n}$. x_i that is at the beginning of $\triangle x$ interval defines the left sum and x_i that is at the end of $\triangle x$

interval defines the right sum. For the left sum $x_i = a + i\triangle x$, $i = 0, 1, ..., n-1$, for the right sum $x_i = a + i\triangle x$, $i = 1, ..., n$

```
[9]: def riemann_right(f, interval, n):
         dx = (interval[1]-interval[0])/n #defining dx
         s = 0 # initial sum equals 0
         for i in range(n):
             s += f.subs(x, interval[0] + (i+1)*dx)*dx  #calculating and adding␣
      ↪f(xi)dx to the sum
         return s
     def riemann_left(f, interval, n):
         dx = (interval[1]-interval[0])/n  #defining dx
         s = 0   # initial sum equals 0
         for i in range(n):
             s += f.subs(x, interval[0] + i*dx)*dx #calculating and adding f(xi)dx to␣
      ↪the sum
         return s

     rsum = riemann_right(f, [0,5],30)
     lsum = riemann_left(f, [0,5], 30)

     display(rsum, lsum)
```

40.8333333333333

39.1666666666667

```
[5]: display(riemann_left(f, [0,5], 300000), riemann_right(f, [0,5],300000))
```

39.9999166666667

40.0000833333333

The Riemann Sum upproaches the true value of the integral $\int_0^5 2x - 3\,dx$ as $n \to \infty$ or alternatively area under the line.

The maximum error of the following method can be estimated by the $\triangle x|f(b) - f(a)| = \frac{b-a}{n}|f(b) - f(a)| = \frac{50}{n}$. That is essentialy the difference between the right and the left sum. We can check it using the results above.

```
[11]: max_err = 50/30
      display(rsum - lsum, max_err)
```

1.66666666666668

```
1.6666666666666667
```

The minimum error will be the actual difference between the true area and the approximated one.

```
[12]: min_err = rsum - 40
      min_err
```

[12]:

```
0.8333333333333336
```

As the function is linear, the slope is constant on the whole interval $[a, b]$, and the error of the upper sum will be the same as that of the left sum.

Proof

From the geometric consideration, the error can be estimated as the sum of triangles $\epsilon = \frac{1}{2}n\triangle x^2 f'(x) = n(\frac{b-a}{n})^2 = \frac{(b-a)^2}{n} = \frac{25}{n}$, what is exactly half of the maximum error.

[15]:

```
calc_err = 25/30
display(min_err, 40-lsum, calc_err) #assesing the difference between the left/
 ↪right sum and the true area
```

```
0.8333333333333336

0.8333333333333343

0.8333333333333334
```

The differences are equal to each other and equal to the calculated above $\epsilon = \frac{25}{30}$.

The image below visualizes left(blue) and right(orange) sums.

[16]:

```
Image(filename="zoom.png", width=200, height=100)
```

[16]: