RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes un informācijas tehnoloģijas fakultāte

Informācijas tehnoloģiju institūts

Studiju darbs mācību priekšmetā
"Ievads operāciju pētīšanā"

Neierobežota nelineāra optimizācija

Izstrādāja: **Artjoms Bogatirjovs**
2. kurss, 7. grupa
Studenta apliecības numurs: **171RDB112**

2022./2023. māc. g.

Saturs

1. Uzdevums

Minimizēt $f(x1, x2) = x_1 - ax_2 + bx_1{}^2 + cx_1 * x_2 + dx_2{}^2$

- Izmantojiet jebkuru programmēšanas valodu pēc savas izvēles
- Izmantojiet konstantu soli **t**
- Veiciet optimizāciju trīs dažādiem sākumpunktiem
- Mainiet soļa **t** vērtības
- Variējiet precizitātes līmeni **ε**
- Veiciet rezultātu analīzi, veidojot diagrammas
  - Katram izvēlētam sākumpunktam atšķirība no optimuma *vs* iterācijas
    (Jūs varat izmantot Wolframalpha, lai izpētītu funkciju un vizualizētu to)
  - Soļa lielums **t** *vs* iterāciju skaits **N**

Varianta izvēle

Variants sakrīt ar studentu apliecības pēdējiem diviem cipariem, no 01 līdz 40.

Ja skaitlis ir

- 00 – tiek izvēlēts variants 40;
- virs 40 – variants tiek izvēlēts kā dalījums pēc moduļa 40
  (piem., 41 atbilst 1,…50 – 10,… 64 – 24,… 72 – 32... 99 - 19)

Studenta apliecības numurs: 171RDB112

Variants	**a**	**b**	**c**	**d**
12	10	5	-1	5

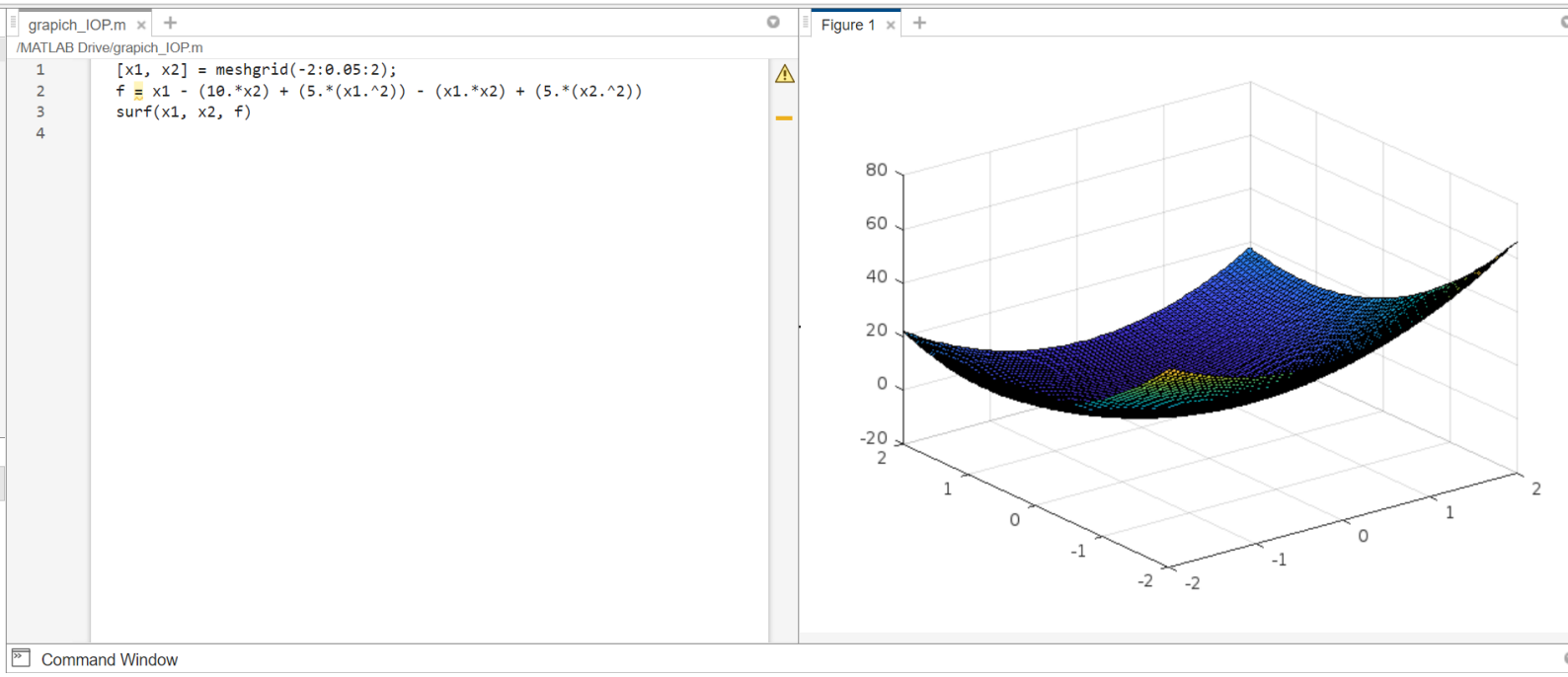
2. Funkcijas vispārēja analīze

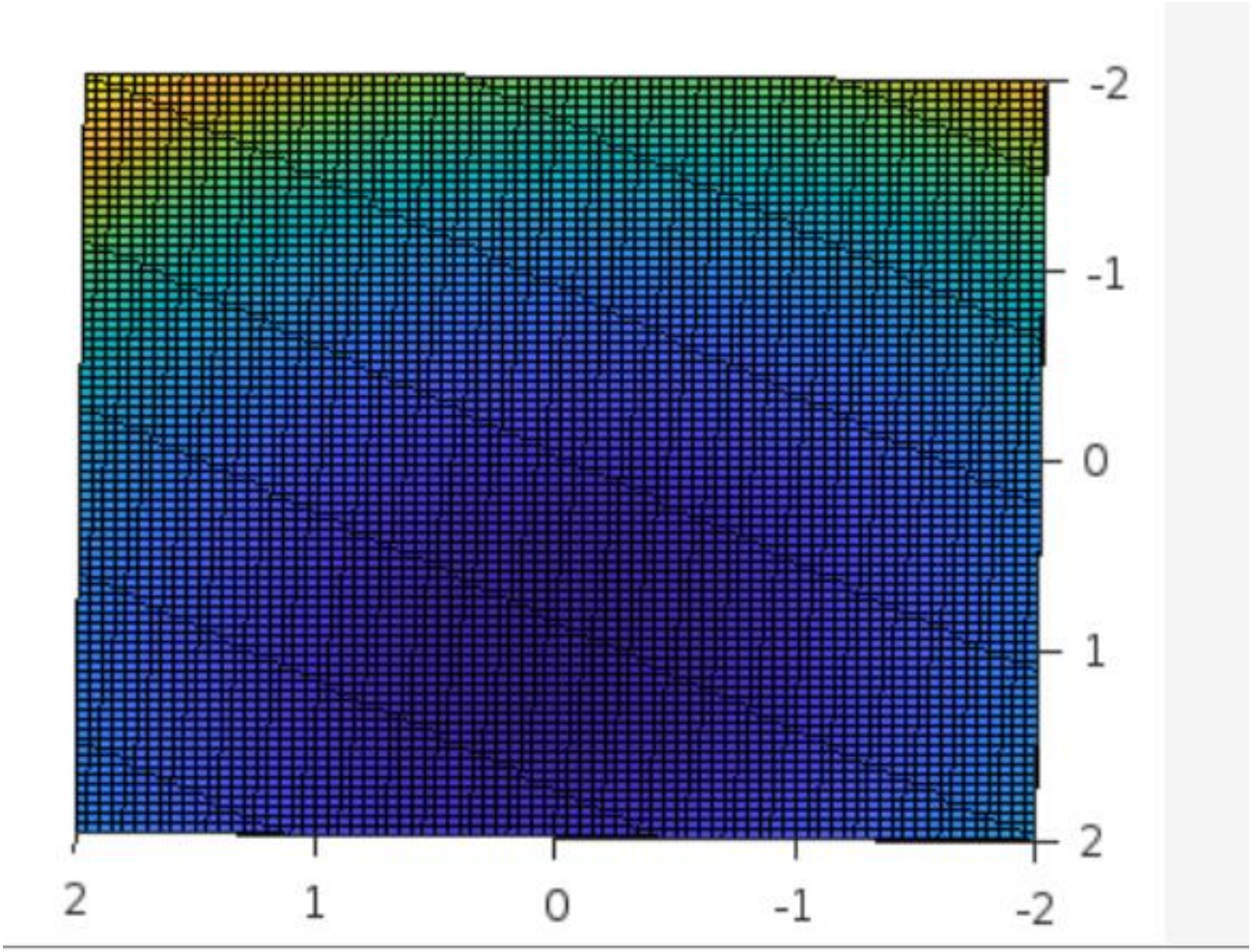
$$f(x_1, x_2) = x_1 - 10x_2 + 5{x_1}^2 - x_1x_2 + 5{x_2}^2 \rightarrow MIN$$

Pirmkārt, lai uzreiz uzzinātu ar ko mēs strādājam, es izanalizēju doto funkciju "Wolfram Alpha" rīkā, un uzzināju globālo minimumu (sk. Pielikums №1). No tā ir redzams, kā globālais minimums atrodas punktā, kur $x_1 = 0$ un $x_2 = 1$. Algoritms, ar kuru "Wolframs Alpha" ir nonācis pie tāda secinājuma, nav zināms, jo tika izmantota tā bezmaksas versija. Ka arī, nav pieejam iespēja apskatīt funkcijas grafiku sīkāk, līdz ar to es izlemju to izdarīt Matlab rīkā, kurā es izveidoju manas funkcijas 3D grafiku ar sekojošam komandām:

```
[x1, x2] = meshgrid(-2:0.05:2);
f = x1 - (10.*x2) + (5.*(x1.^2)) - (x1.*x2) + (5.*(x2.^2))
surf(x1, x2, f)
```

No tā tik saņemts sekojošs grafiks:





Ņemot vērā, ka tumši zila krāsa atspoguļo mazākas funkcijas vērtības, tad ir skaidri redzams, ka funkcijas minimums apmēram punktā, kur $x_1 = 0$ un $x_2 = 1$. Tātad mēs varam uzsākt mūsu funkcijas optimizēšanu izmantojot gradienta metodi, bet jau zinot, ka optimums ir apmēram punktā **(0; 1)**, skatoties pēc "Wolfram Alpha" rezultātiem.

3. Optimizācija ar gradienta metodi

Pirmkārt, risinot šo uzdevumu, tika saprasta gradienta metodes būtība. Šo metodi izmanto, lai atrisinātu nelineāras funkcijas ar 2 vai vairāk mainīgajiem. Būtība ir tāda, ka katru iterāciju algoritms izmaina mainīgo pašreizējas vērtības gradienta virziena, jo funkcijas gradients ir visstraujākās izmaiņas virziens. Algoritms var strādāt bezgalīgi, atkārtojot sekojošas iterācijas. Šeit es aprakstīšu algoritmu tieši divu mainīgai funkcijas optimizācijai:

1. Rēķinam funkcijas gradientu punktam, kur pašlaik esam (vai sākumpunktam, ja 1. iterācija)
2. Pārvietojamies uz nākamo punktu, pēc formulas $\boldsymbol{x = x - t * grad(x)}$, kur $\boldsymbol{t}$ ir algoritma koeficients, kuru var rēķināt ar funkcijas parciālajiem atvasinājumiem, lai dabūtu optimālo, bet arī var izmantot konstantu, ka arī darījām mēs uzdevumā.
3. Pārbaudām nobeigšanas nosacījumu. Mana gadījumu, es pārbaudīju, vai izpildās vismaz viens nosacījums, lai pabeigtu darbības:
   a. Ja funkcijas izmaiņas vienā solī norma (otra norma) ir mazāka par noteiktu dotu precizitāti (***eps***). Tas liecina par to, ka funkcija konverģē uz optimumu, un nepieciešama precizitāte ir sasniegta.
   b. Ja iterāciju skaits ir lielāks par **50**. Visbiežāk tas liecina par to, ka funkcija diverģē, un algoritma darbības arī vajag pabeigt.
   c. Ir vēl viena iespēja, kuru es neizmantoju. Ja optimums jau ir zināms, vai rēķināt vienādojuma nesaisti (***norm(x_current - x_optimal)***), un salīdzināt ar noteiktu, nepieciešamu precizitāti.

Izmantojot iepriekš doto pirmkodu, es pārbaudīju visus nepieciešamus algoritma parametrus:

- Sākumpunktu ***x***, ņemot vērā, ka optimums ir pie **(0; 1)**, es izmantoju sekojošus sākumpunktus: **(-1, -1)**, **(0, 0)** un **(1, 1).**
- Precizitāti ***eps***, ar kuru es pārbaudīju funkcijas konverģenci. Es izmantoju sekojošas precizitātes: **0.1**, **0.01** un **0,001**
- Soli ***t***, tam es piešķiru ļoti daudz dažādu vērtību, kurus es paskaidrošu vēlāk.

4. Programmas pirmkods

Programmas kodu es uztaisīju Matlaba Online. Taču vēl izmantoju MS Excel priekš vizualizācijas. Tas kods bija izmantots rēķinot funkcijas minimumu pie $\text{eps} = 0.001$ un $x_{\text{sakuma}} = [1; 1]$. Tas pats kods, bet ar eps, t un x_{sakuma} izmainām tika izmantots arī citiem rēķiniem.

```
clc; clear;
%Dota funkcija
syms f(x1, x2)
f(x1, x2) = x1 - (10.*x2) + (5.*(x1.^2)) - (x1.*x2) + (5.*(x2.^2));
%Funkcijas abi parciālie atvasinājumi
dif_x1(x1, x2) = diff(f, x1);
dif_x2(x1, x2) = diff(f, x2);
%Pie šis vērtības  visticamāk vairs nekonverģē
will_not_converge = 50;
%Sakuma punkts
x_start = -1
%Visas testējamās T vērtības. Bus paskaidrots protokola
t = [1 0.95 0.9 0.85 0.8 0.75 0.7 0.65 0.6 0.55 0.5 0.45 0.4 0.35
0.3 0.25 0.2 0.15 0.1 0.05];
%Funkcijas izmainās viena soli robeža
eps = 0.001;
%Masīvs prieks rezultātu atspoguļošanas
data = [];
%Cikls prieks katras t vērtības
for i = 1:size(t, 2)
    %Iterācijas numurs
    iter = 1;
    %Pašreizējas x1 un x2 vērtības
    x = [x_start, x_start];
    %Pagājušas x1 un x2 vērtības (lai rēķinātu starpību)
    x_last = [1000000, 1000000];
    %Funkcijas gradients
    grad_x =[0, 0];
    %Izmantojam gradienta meklēšanas metodi, kamēr:
    % Funkcijas vērtības izmaina ir mazākā par EPS (vērtības
konverģē)
    % Iterвciju skaits ir > 'will_not_converge'
    while ( norm(x-x_last) > eps) && (iter < will_not_converge)
        grad_x(1) = dif_x1(x(1), x(2));
        grad_x(2) = dif_x2(x(1), x(2));
        x_last(1) = x(1);
        x_last(2) = x(2);
        x(1) = x(1) - t(i)*grad_x(1);
        x(2) = x(2) - t(i)*grad_x(2);
        iter = iter + 1;
    end
    % Nepiecienāms iterвciju skaits pie dota t
    % (ja ir vienāds ar 'will_not_converge' - diverģē)
    data=[data; t(i) iter];
end
%Paradīt rezultātus masīv rindu veida
data = data.'
```

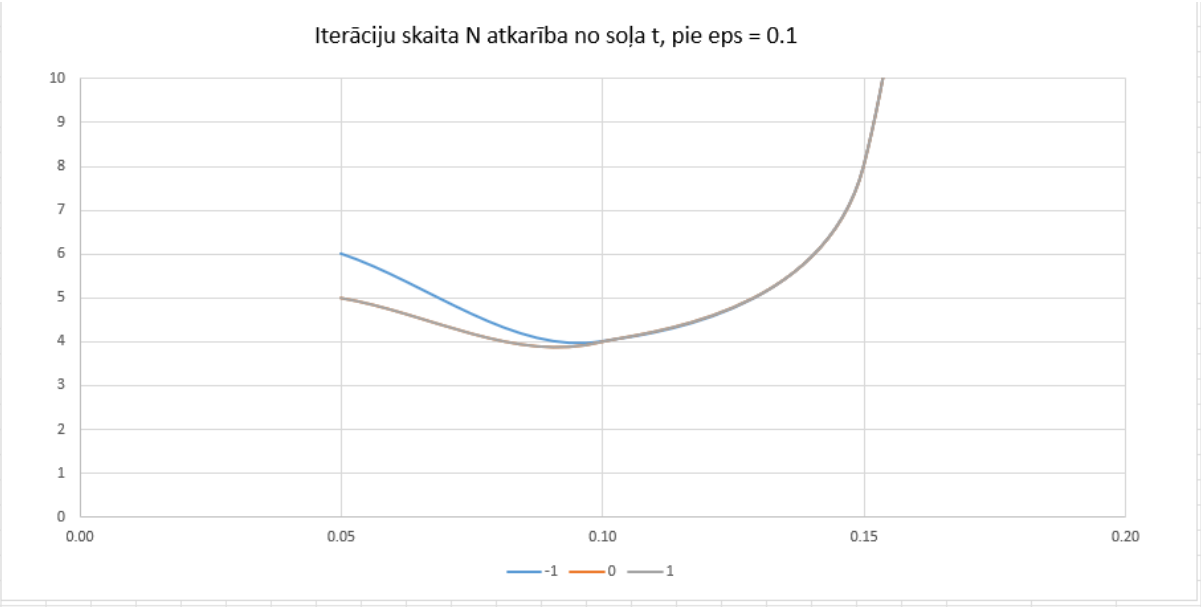
5. Rezultātu analīze

Izmantojot tikko paradīto kodu, es pārbaudīju iterāciju skaitus pie daudziem soļa t vērtībām visiem noradītām sākumpunktiem pie precizitātes līmeni **eps = 0.1**. Zemāk Jūs variet apskatīt tabulu ar iegūtiem datiem, kur pirmā kolonā ir ***x*** sākumvērtības, bet pirmā rinda ir ***t*** vērtības.

	1.00	0.95	0.90	0.85	0.80	0.75	0.70	0.65	0.60	0.55	0.50	0.45	0.40	0.35	0.30	0.25	0.20	0.15	0.10	0.05
-1	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	8	4	6
0	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	8	4	5
1	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	8	4	5

Pirmā lieta, kuru var redzēt no tabulas, ir tas, ka vairāk nekā puse no visam vērtībām ir **"50"**: tas nozīmē, ka pie dota soļa ***t*** un pie dota sākumpunkta x_0 funkcija diverģē, un nevar sasniegt optimumu. No tā var secināt, ka ***t*** vērtības lielākas vai vienādas ar **0.2** nedod mums rezultātu. Visas pārējās vērtības ļauj mums sasniegt rezultātu ar mazāku iterāciju skaitu. Zemāk var apskatīt grafiku izveidotu no tabulas:

Ir skaidri redzams, ka grafika labā puse, kur funkcija diverģē, un iterāciju skaiti ir 50, ir pilnīgi bezjēdzīga un tikai traucē adekvāti vizualizēt iegūtus datus. Tāpēc tika nolemts izveidot vēl vienu grafiku ar soļa ***t*** vērtībām mazākam par **0.2**:

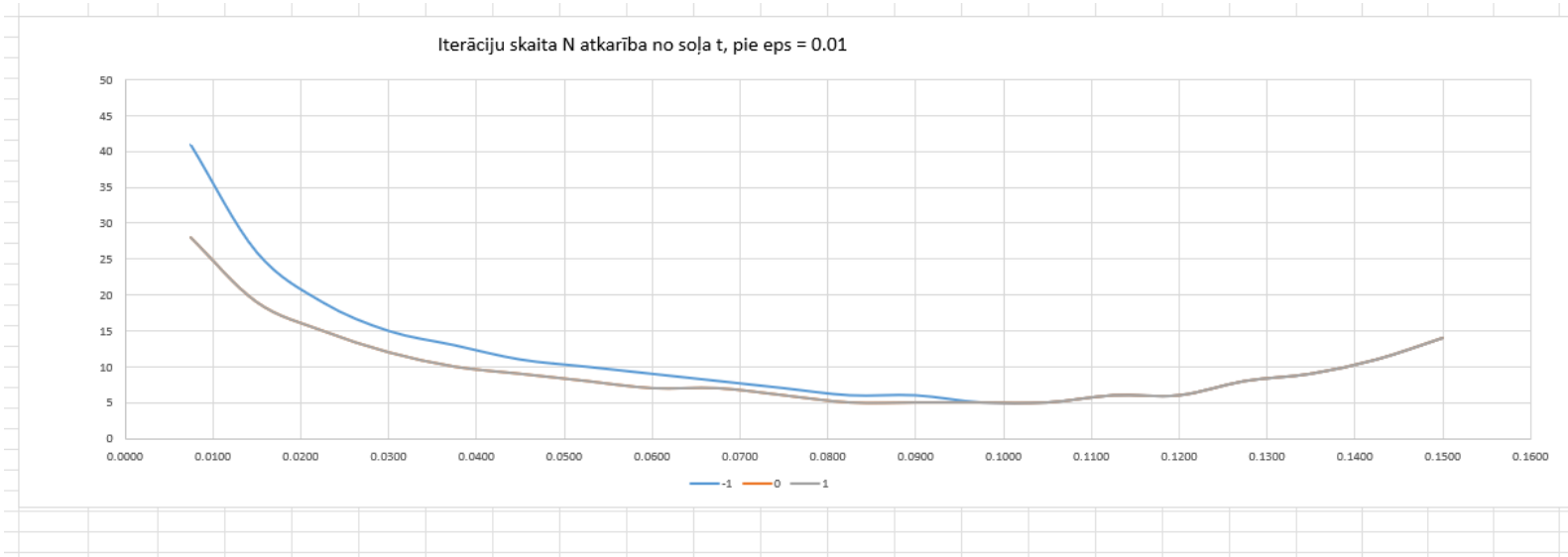


Ir skaidri redzams, ka vislabākais sākuma punkts ir (0; 0) un (1; 1), tāpēc ka tas ļauj optimizēt iterāciju skaitu gandrīz ar visām ***t*** vērtībām, un pie tam, pieder vismazākais iterāciju skaits eksperimentā – **4** iterācijas, lai sasniegtu precizitāti eps = 0.1. Citam punktam(-1; -1) ir sliktākie rezultāti – tie konverģe lēnāk. Visticamāk tas notiek tieši tā, jo sis punkts ir tālāk no optimuma, kas ir **(0; 1)**.

Tagad var pārbaudīt to pašu algoritmu, bet izmantojot precizitātes līmeni **eps = 0.01**, kas ir desmit reizēs mazāku.

	0.1500	0.1425	0.1350	0.1275	0.1200	0.1125	0.1050	0.0975	0.0900	0.0825	0.0750	0.0675	0.0600	0.0525	0.0450	0.0375	0.0300	0.0225	0.0150	0.0075
-1	14	11	9	8	6	6	5	5	6	6	7	8	9	10	11	13	15	19	26	41
0	14	11	9	8	6	6	5	5	5	5	6	7	7	8	9	10	12	15	19	28
1	14	11	9	8	6	6	5	5	5	5	6	7	7	8	9	10	12	15	19	28

Šajā gadījumā tika izmainīts soļa t diapazona, sakot ar 0.0075 un ar 0.0075 soliem ejot uz 0.15. Šajā konfiguracijā ir redzams, ka visas vertības dod rezultātu un funkcija pie tiem konverģē.



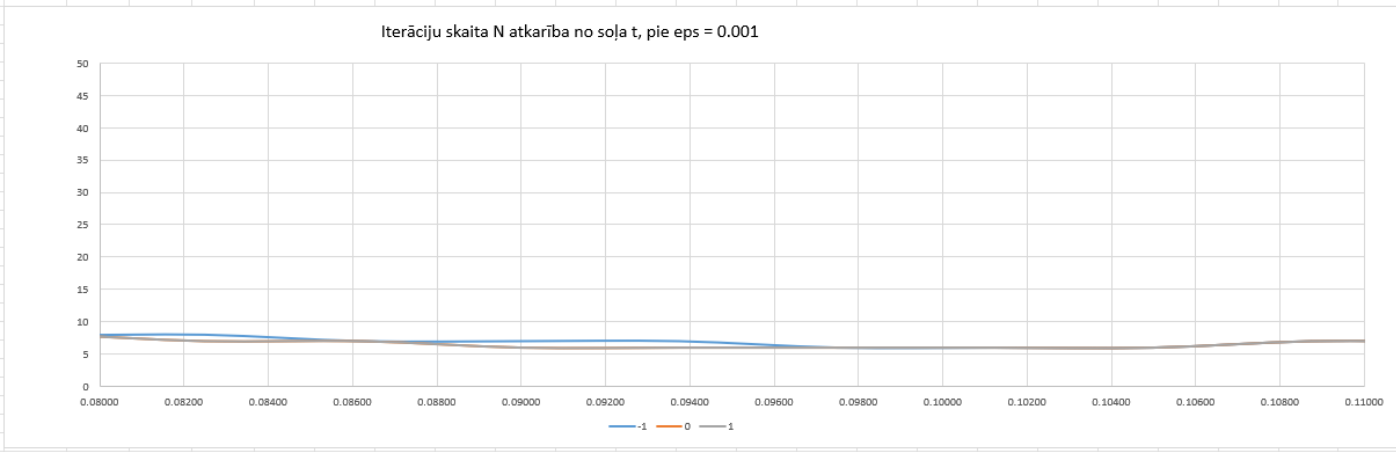
No grafika var ļoti labi redzēt, ka pie ***t*** vērtībām mazākām par 0.02 un lielākām par 0.12 iterāciju skaits saka strauju palielinies, bet vismazākais iterāciju skaits (pie eps = 0.1) ir joprojām pie sākumpunkta (0; 0) un ir vienāds ar 5.

Pēdējais apskatītais gadījums ir pie precizitātes līmeņa eps = 0.001, kas ir vēl desmit reizēs mazākas. Nomainot koda ***eps*** un ***t*** vērtības, tika dabūts sekojošs grafiks:

	0.15000	0.14625	0.14250	0.13875	0.13500	0.13125	0.12750	0.12375	0.12000	0.11625	0.11250	0.10875	0.10500	0.10125	0.09750	0.09375	0.09000	0.08625	0.08250	0.07875
-1	19	17	15	13	12	11	10	9	9	8	7	7	6	6	6	7	7	7	8	8
0	19	17	15	13	12	11	10	9	9	8	7	7	6	6	6	6	6	7	7	8
1	19	17	15	13	12	11	10	9	9	8	7	7	6	6	6	6	6	7	7	8

0.07500	0.07125	0.06750	0.06375	0.06000	0.05625	0.05250	0.04875	0.04500	0.04125	0.03750	0.03375	0.03000	0.02625	0.02250	0.01875	0.01500	0.01125	0.00750	0.00375
9	10	10	11	12	12	13	14	16	17	18	20	23	26	29	34	42	50	50	50
8	8	9	10	10	11	12	12	13	15	16	17	19	22	25	29	34	43	50	50
8	8	9	10	10	11	12	12	13	15	16	17	19	22	25	29	34	43	50	50

No tabulas un grafikiem atkal var redzēt, ka pie ***t*** vērtībām mazākām par ≈0.03 un lielākām par 0.12 iterāciju skaits saka strauju palielinies, bet vismazākais iterāciju skaits (pie eps = 0.1) ir joprojām pie sākumpunkta (0; 0) un ir vienāds ar 6. Šī vērtība ir no t=0.105 līdz t=0.09, līdz ar to, pēc intereses dēļ, var pārbaudīt šo diapazonu ar vēl mazāko ***t*** soļu (Δt = 0.002) vērtībām:



No grafika droši vien var redzēt, ka nekas nav mainījušies, un sākumpunkts (0; 0) un (1; 1) joprojām ir visoptimāls ar 6 iterācijām pie eps = 0.001.

6. Secinājumi

Darba gaitā, izmantojot Matlab programmatūru, es optimizēju (minimizēju) funkciju $f(x_1, x_2) = x_1 - 10x_2 + 5{x_1}^2 - x_1x_2 + 5{x_2}^2$, izmantojot gradienta metodi ar dažādiem parametriem. Bija izmantoti 3 sākuma punkti: (-1; -1), (0; 0) un (1, 1), kuri bija izvēlēti pēc optimāla punkta (0; 1) atrašanas ar "Wolfram Alpha" palīdzību. Kā arī darbā bija izmantoti 3 dažādi precizitātes līmeni eps = {0.1; 0.01; 0.001}.

Bija izmantoti daudz dažādu soļu ***t*** vērtību, sakot no 0.01 un beidzot ar 1, bet, ka bija noteikts, tomēr ir noteiktais intervāls ar ***t*** vērtībām, pie kurām funkcija konverģē, un tas intervāls ir apmērām (0.09 : 0.1). Tas ir iespējams tikai pateicoties tam, ka bija izmantots konstants solis ***t***, bet, ja solis ***t*** būtu rēķināts no jaunā pēc katras iterācijas, izmantojot funkcijas parciālus atvasinājumus, tad funkcija konverģētu daudz ātrāk.

Galu galā, varu secināt, ka Gradienta metode ir ļoti noderīga, kad mēs runājam par nelineāras funkcijas neierobežotu optimizāciju. Viena no priekšrokām ir tas, ka metode paliek ne īpaši daudz sarežģītākā, ja mēs strādājam ar vairākiem mainīgajiem. Metodes būtība ir ļoti vienkāršā, un ja ir runa tikai par dažām iterācijām, tad var vispār bez skaitļošanas programmatūras palīdzības visu aprēķināt, bet, ja ir nepieciešama liela precizitāte, kā mūsu gadījuma, tad labāk izmantot palīg rikas, tādas ka Matlab.

7. Izmantotās literatūras un avotu saraksts

[1] Nelineāras programmēšanas praktiskā nodarbība 20.03 - 27.03. Ortus.

[2] https://matlab.mathworks.com/

[3] https://www.wolframalpha.com/input?i2d=true&i=minimize+x1-10*x2%2B5*Power%5Bx1%2C2%5D-x1*x2%2B5*Power%5Bx2%2C2%5D

[4] https://github.com/ArtjomsBogatirjovs/IOP_2darbs

8. Pielikumi

Pielikums: “Wolfram Alpha” risinājums

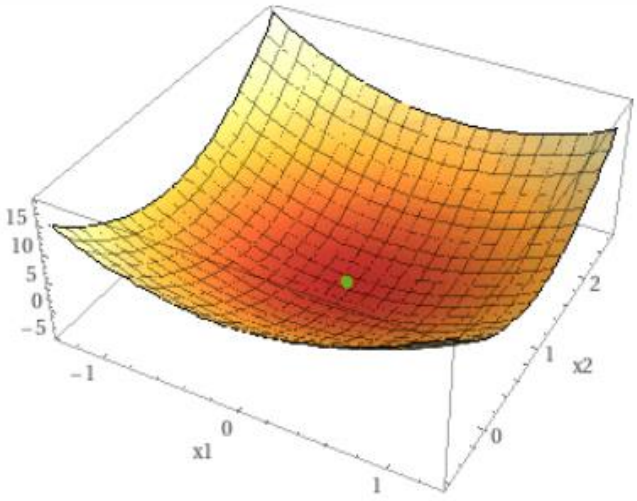
Input interpretation

minimize	$x1 - 10\,x2 + 5\,x1^2 - x1\,x2 + 5\,x2^2$

Global minimum

$\min\left\{x1 - 10\,x2 + 5\,x1^2 - x1\,x2 + 5\,x2^2\right\} = -5$ at $(x1, x2) = (0, 1)$

3D plot



Contour plot