Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет
Кафедра прикладной математики

Численные методы

Практическая работа №4

Факультет: прикладной математики и информатики
Группа: ПМ-63
Студент: Кожекин М.В.
Преподаватели: Задорожный А. Г.
Персова М.Г.

Новосибирск
2018

1. Цель работы

Разработать программу решения системы нелинейных уравнений (СНУ) методом Ньютона. Провести исследования метода для нескольких систем размерности от 2 до 10.

2. Задание

2. m ≥ n. Для нахождения Δx^k из системы (4.2) те её (m – n) уравнений, для которых абсолютные значения $F_i(x^k)$ минимальны, исключаются из системы. При вычислении нормы вектора F^k в процессе подбора параметра β^k учитывать все уравнения системы. Производные при формировании матрицы Якоби вычислять аналитически.

3. m ≥ n. Для нахождения Δx^k из системы (4.2) те её (m – n + 1) уравнений, для которых абсолютные значения $F_i(x^k)$ минимальны, проводится свертка. Процедура свертки заключается в следующем. Вместо исключаемых уравнений берется уравнение, получающееся возведением в квадрат исключаемых уравнений и их сложением.

7. Производные при формировании матрицы Якоби вычислять аналитически.

$$F_i \approx F_i(x^k) + \sum_{j=1}^{n} \left.\frac{\partial(F_i(x))}{\partial x_j}\right|_{x=x^k} \Delta x_j^k, i = \overline{1,m}$$

или в матричном виде:

$$A^k \Delta x^k = -F^k$$

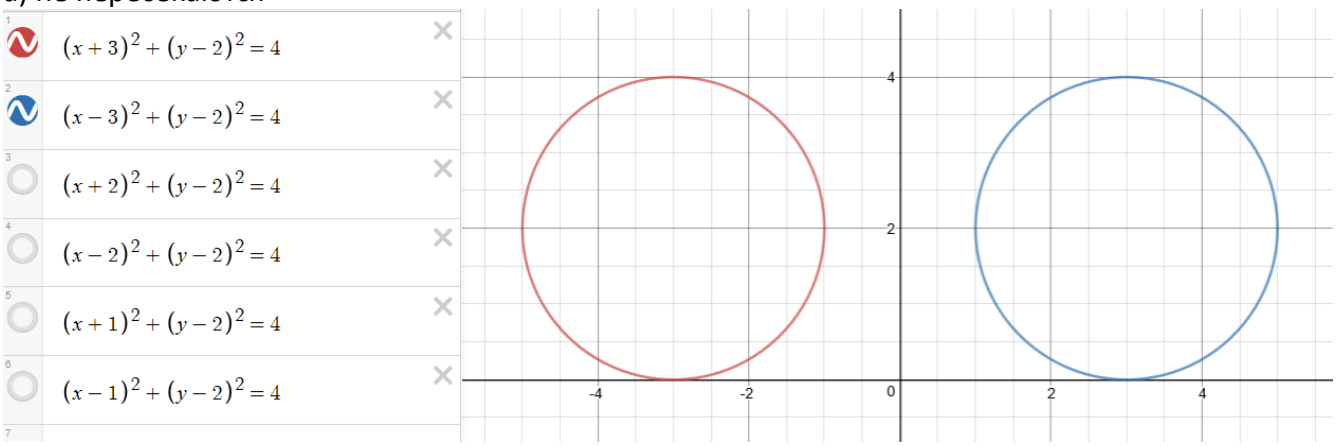
где F^k – значение вектор-функции F при $x = x^k$; A – матрица Якоби:

$$A_{ij}^k = \left.\frac{\partial(F_i(x))}{\partial x_j}\right|_{x=x^k}$$

3. Исследования

- **Тест 1 (2 окружности)**

а) не пересекаются



- Начальное приближение не лежит на оси симметрии (1, 1)

k	x	y	beta	Discrapancy
0	0	3.5	1	10.25304832720494
1	0	1.083333333333333	1	8.259400041359534
2	0	2.676136363636364	0.5	7.717590224596797
3	0	1.667249570282659	0.25	7.227653565846338
4	0	2.147219165365004	0.0625	7.10171874497405
5	0	1.880733317643066	0.015625	7.091184351402205
6	0	2.044960361007079	0.0078125	7.073926551331238
7	0	1.990637103283104	0.0009765625	7.071191787249764
8	0	2.006934470489725	6.103515625e-05	7.07113581698472
9	0	1.995932234749719	3.0517578125e-05	7.071091212453013
10	0	2.000621185137776	7.62939453125e-06	7.071068357569443
11	0	1.99966165398111	2.384185791015625e-07	7.071067973761856
12	0	2.000542477794784	5.960464477539063e-08	7.071068228043294

- Начальное приближение лежит на оси, соединяющей центы окружностей (0.5, 2)

Не вычисляется ни одним из методов, потому что матрица Якоби вырожденная.

- Начальное приближение лежит на оси, перпендикулярной оси, соединяющей центры окружностей и пересекающей ее на равных расстояниях от центров окружностей (0, 4)

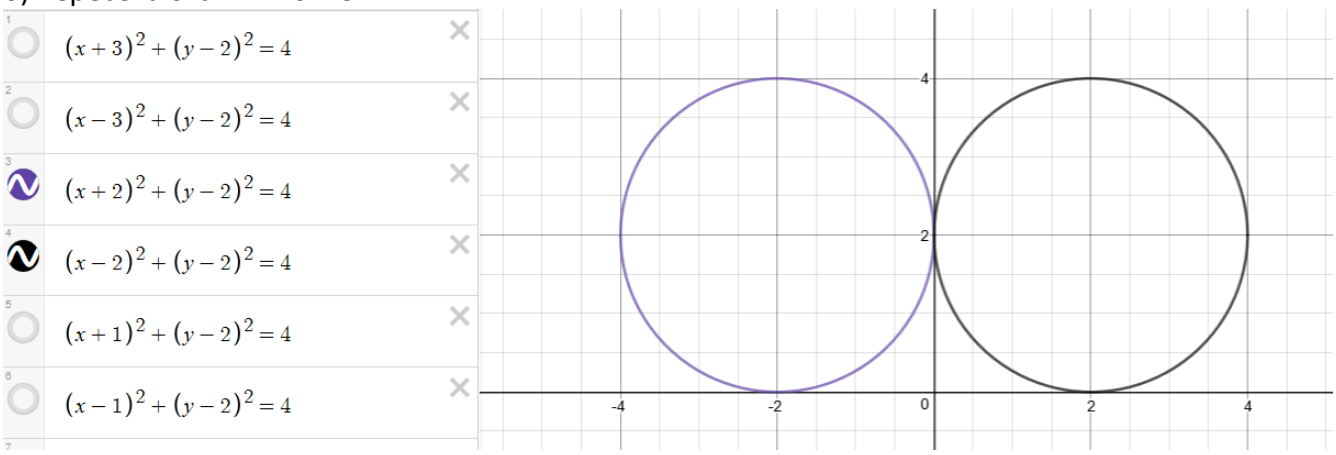
k	x	y	beta	Discrapancy
0	0	1.75	1	7.159456159513794
1	0	2.06640625	0.03125	7.077304196745935
2	0	1.992811988381779	0.001953125	7.071140880760298
3	0	2.00342615231868	3.0517578125e-05	7.071084412639252
4	0	1.997859111121173	7.62939453125e-06	7.071074293779259
5	0	2.000086399048287	1.9073486328125e-06	7.07106782242229
6	0	1.996637018178596	5.960464477539063e-08	7.071083806121266

- Начальное приближение лежит в центре или внутри одной из окружностей (3, 1)

k	x	y	beta	Discrapancy
0	0	-0.5	1	15.90990257669732
1	0	1.75	1	7.159456159513794
2	0	2.06640625	0.03125	7.077304196745935
3	0	1.992811988381779	0.001953125	7.071140880760298
4	0	2.00342615231868	3.0517578125e-05	7.071084412639252
5	0	1.997859111121173	7.62939453125e-06	7.071074293779259
6	0	2.000086399048287	1.9073486328125e-06	7.07106782242229
7	0	1.996637018178596	5.960464477539063e-08	7.071083806121266

Если же выберем точку в центр одной из окружностей, то решения не бужет найдено. Метод не работает если начальное приближение лежит на оси, соединяющей центы окружностей потому что матрица Якоби вырожденная.

б) пересекаются в 1 точке



- Начальное приближение не лежит на оси симметрии (1, 1)

k	x	y	beta	Discrapancy
0	0	1	1	1.414213562373095
1	0	1.5	1	0.3535533905932738
2	0	1.75	1	0.08838834764831845
3	0	1.875	1	0.02209708691207961
4	0	1.9375	1	0.005524271728019903
5	0	1.96875	1	0.001381067932004976
6	0	1.984375	1	0.0003452669830012439
7	0	1.9921875	1	8.631674575031098e-05
8	0	1.99609375	1	2.157918643757775e-05
9	0	1.998046875	1	5.394796609394436e-06
10	0	1.9990234375	1	1.348699152348609e-06
11	0	1.99951171875	1	3.371747880871523e-07

- Начальное приближение лежит на оси, соединяющей центы окружностей (0.5, 2)

Не вычисляется ни одним из методов, потому что матрица Якоби вырожденная.

- Начальное приближение лежит на оси, перпендикулярной оси, соединяющей центры окружностей и пересекающей ее на равных расстояниях от центров окружностей (0, 4)

k	x	y	beta	Discrapancy
0	0	3	1	1.414213562373095
1	0	2.5	1	0.3535533905932738
2	0	2.25	1	0.08838834764831845
3	0	2.125	1	0.02209708691207961
4	0	2.0625	1	0.005524271728019903
5	0	2.03125	1	0.001381067932004976
6	0	2.015625	1	0.0003452669830012439
7	0	2.0078125	1	8.631674575031098e-05
8	0	2.00390625	1	2.157918643757775e-05
9	0	2.001953125	1	5.394796609394436e-06
10	0	2.0009765625	1	1.348699152348609e-06
11	0	2.00048828125	1	3.371747880871523e-07

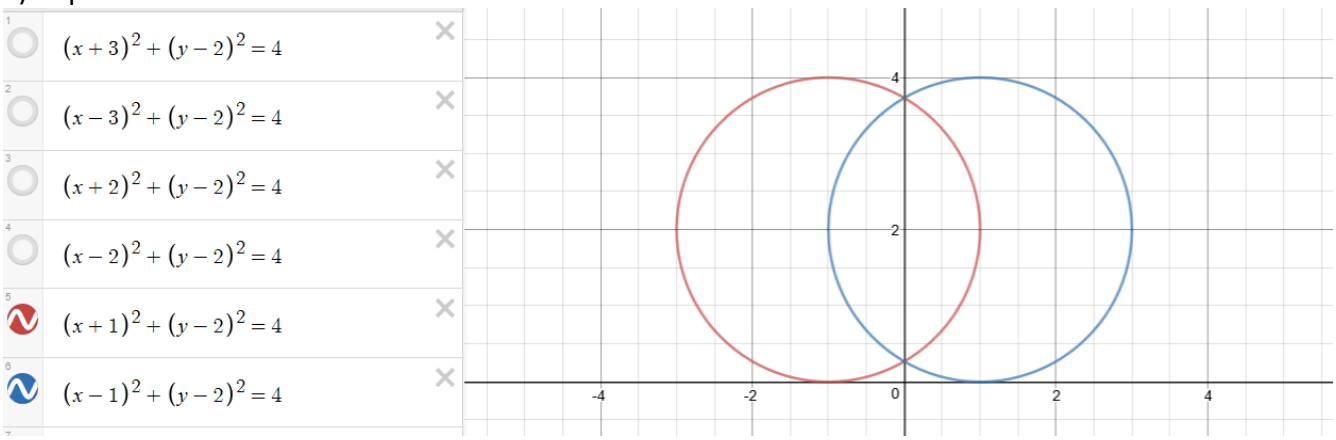
- Начальное приближение лежит в центре или внутри одной из окружностей (3, 1)

k	x	y	beta	Discrapancy
0	1.5	-1	0.5	18.03122292025696
1	0	0.125	1	4.971844555217912

2	0	1.0625	1	1.242961138804478
3	0	1.53125	1	0.3107402847011195
4	0	1.765625	1	0.07768507117527988
5	0	1.8828125	1	0.01942126779381997
6	0	1.94140625	1	0.004855316948454993
7	0	1.970703125	1	0.001213829237113748
8	0	1.9853515625	1	0.000303457309278437
9	0	1.99267578125	1	7.586432731960926e-05
10	0	1.996337890625	1	1.896608182990232e-05
11	0	1.9981689453125	1	4.741520457475579e-06
12	0	1.99908447265625	1	1.185380114368895e-06

Если же выберем точку в центр одной из окружностей, то решения нет, т.к. метод не работает если начальное приближение лежит на оси, соединяющей центы окружностей потому что матрица Якоби вырожденная.

в) пересекаются в 2 точках



- Начальное приближение не лежит на оси симметрии (1, 1)

k	x	y	beta	Discrapancy
0	0.5	0.25	0.5	1.481658698891212
1	0	0.1964285714285714	1	0.3576120392097785
2	0	0.2665311173974541	1	0.006949964393034037
3	0	0.2679486123985588	1	2.841568106923969e-06
4	0	0.2679491924310256	1	4.760520334729206e-13

- Начальное приближение лежит на оси, соединяющей центы окружностей (0.5, 2)

Не вычисляется ни одним из методов, потому что матрица Якоби вырожденная.

- Начальное приближение лежит на оси, перпендикулярной оси, соединяющей центры окружностей и пересекающей ее на равных расстояниях от центров окружностей (0, 4)

k	x	y	beta	Discrapancy
0	0	3.75	1	0.08838834764831845
1	0	3.732142857142857	1	0.0004509609573892773
2	0	3.732050810014728	1	1.198217190311093e-08

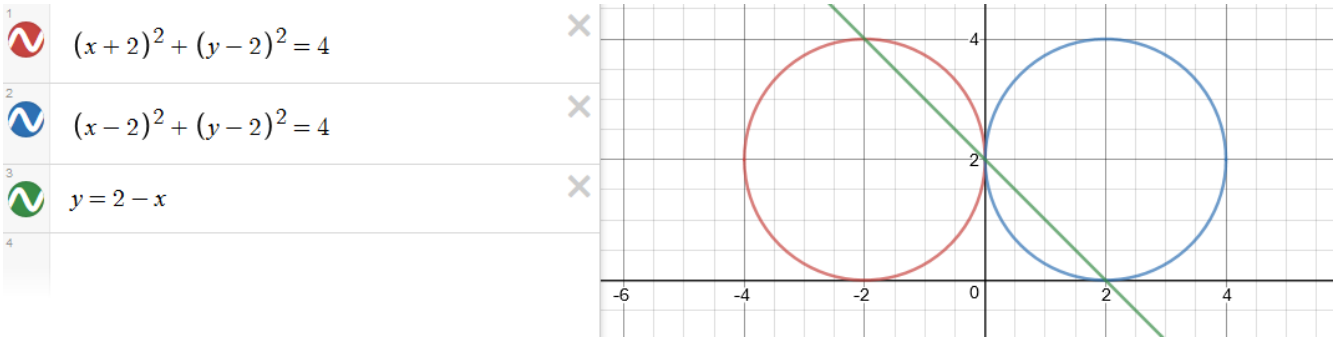
- Начальное приближение лежит в центре или внутри одной из окружностей (3, 1)

k	x	y	beta	Discrapancy
0	2.25	-0.375	0.25	12.61650781838025
1	0	-0.8848684210526317	1	7.527103329224815

2	0	0.03761139950789161	1	1.203451928311906
3	0	0.2544311005416161	1	0.06648328655181131
4	0	0.2678968488059615	1	0.0002564342206054525
5	0	0.2679491916402186	1	3.874623926581142e-09

Если же выберем точку в центре одной из окружностей, то решения нет, т.к. метод не работает если начальное приближение лежит на оси, соединяющей центы окружностей потому что матрица Якоби вырожденная.

- **Тест 2 (2 окружности и прямая)**



  - Начальное приближение (1, 4)

2 метод

k	x	y	beta	Discrapancy
0	0.625	4.28125	0.125	9.140471523985299
1	0	3.226241438356165	1	2.454730813225014
2	0	2.613120719178082	1	0.8115073767695261
3	0.04364469378078339	2.448929365938366	1	0.6215805260695239
4	0.03847147983700082	2.051453933670516	1	0.2355470353849267
5	9.71445146547012e-17	2.04010929540829	1	0.04017376964102233
6	-0.0004829093769750173	1.995974973800161	1	0.005271097631777001
7	-0.0004349780286404151	2.002401920390958	0.125	0.003150162153793625
8	-0.0004349779263974128	2.002401962219357	5.960464477539063e-08	0.003150187884198415

3 метод

k	x	y	beta	Discrapancy
0	0.2000000000000001	2.95	1	2.092632910952135
1	-0.1566399256960588	2.858797722664557	1	1.561947447487779
2	0.04019995713312322	2.544635386900305	1	0.7560715625479764
3	0.09585837483613649	2.341982258850684	0.5	0.7194232873675606
4	0.04004340735683657	2.497790028292719	0.25	0.6818908257543161
5	0.07181787077785401	2.402187037995985	0.0625	0.6674202888689231
6	0.08022668573615416	1.917631606676194	1	0.454220701013152
7	0.04987493208042018	2.165219004346736	0.5	0.3572673462213245
8	0.01187803116390639	2.289706835662327	0.5	0.3310651725140578
9	0.03974357068273535	2.176856420074269	0.03125	0.315627096291381
10	0.02546345520650619	1.827676058454164	1	0.2101378406206743
11	0.02397510852448222	2.039482307331721	0.5	0.1497656006698614
12	0.001877020031085787	1.930799064308337	1	0.06849222067444202
13	0.001877019313812545	1.93079903894068	5.960464477539063e-08	0.06849224617634536

7 метод

k	x	y	beta	Discrapancy
0	-0.1315788142344698	3.44736864275618	1	3.347835551871922

1	0.07296240535491941	2.837119010971346	1	1.412703687756589
2	-0.02492866492713512	2.483470428052274	1	0.5830956774088416
3	-0.08102012822116861	2.310291009815785	0.5	0.5327059123428594
4	-0.08153410826306209	2.306759943150293	0.0001220703125	0.5326889669052823
5	-0.08766785056348504	2.243395570521387	0.03125	0.5283469022047482
6	-0.03978172909241681	1.796268645704185	1	0.3371271353467308
7	-0.024817164008585	2.024817164028618	1	0.1403978871697387
8	-0.02364904779306076	1.993572581480298	0.0625	0.1371210925027832
9	-0.01162513491362172	1.910643452507044	0.5	0.1210527190710512
10	-0.006387601299145373	2.00638760130629	1	0.0361339138412354
11	-0.006350134102530448	2.002481693899149	0.0078125	0.03612953986409145
12	-0.004789982406210556	1.982069247214644	0.25	0.03536488249289837
13	-0.002397596533030997	2.0023975965349	1	0.01356286388214754
14	-0.00239583977917737	2.001909417387025	0.0009765625	0.013561649120012
15	-0.002323479351449951	1.99929545219588	0.03125	0.01348787675830626
16	-0.001159686347900765	1.991715873256389	0.5	0.01149918759657359
17	-0.0005802701170504256	2.000580270117699	1	0.003282503615647966
18	-0.000580243551824625	2.000549778787996	6.103515625e-05	0.003282494700714044
19	-0.0005791511852936769	2.000401065147423	0.001953125	0.003281010542976241
20	-0.0005431479639263006	1.999449461999455	0.0625	0.003261358681177242
21	5.968905454911428e-07	1.997286353327218	1	0.002713071870631931
22	8.776678452129862e-06	1.999991223321769	1	4.96483907987583e-05
23	8.776677667379142e-06	1.999991286515663	5.960464477539063e-08	4.964842657624202e-05

- Начальное приближение (-1, 1)

2 метод

k	x	y	beta	Discrapancy
0	-0.5	0.25	0.25	5.916740023019433
1	0.1851851851851851	0.03009259259259259	0.5	5.910532575870967
2	-2.775557561562891e-17	1.006341939765853	1	1.713797370631672
3	4.049217880932177e-08	1.006341917491255	5.960464477539063e-08	1.713797411075156

3 метод

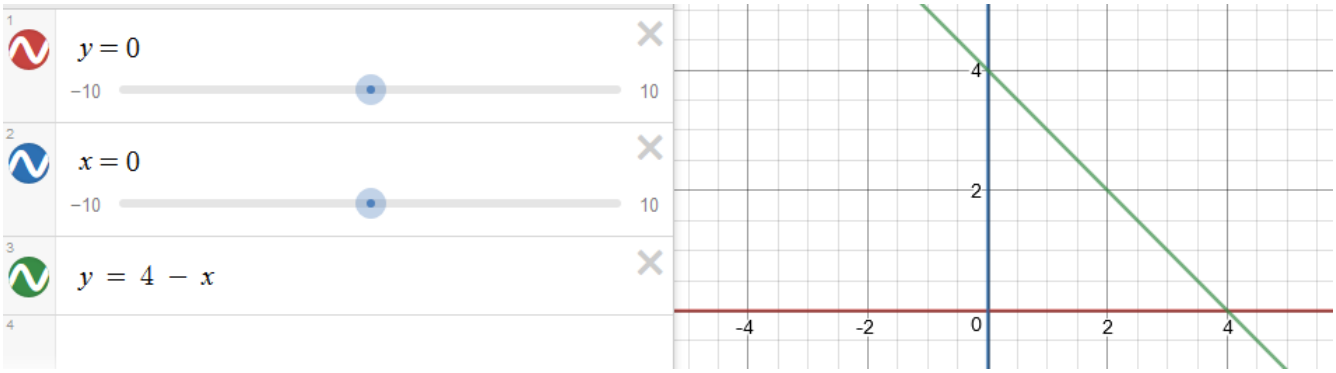
k	x	y	beta	Discrapancy
0	-0.1111111111111112	1.333333333333333	1	1.190510156939175
1	0.008681405340729878	1.291953327532133	0.5	0.9971651715863056
2	0.04917186421820864	1.30592979824195	0.03125	0.9808520648065188
3	0.1122654547886076	1.507534356541397	0.03125	0.8234340964550345
4	-0.06385908798441352	1.467068778963787	1	0.8078675989941718
5	-0.06089234882488446	1.460613395219385	0.015625	0.8078485025369191
6	-0.05487144901831988	1.448662680951941	0.03125	0.8076602644342091
7	-0.04820469450611442	1.437281911221052	0.03125	0.8068923995957026
8	-0.03333034374577527	1.416143923287257	0.0625	0.806470492499162
9	-0.01368180238232644	1.399327678801252	0.0625	0.8025304230192077
10	0.01701968743008963	1.392737319277438	0.0625	0.7937652654551245
11	0.06849117417369108	1.427905362220628	0.015625	0.7900351904739487
12	0.06849113418050706	1.427905280751184	5.960464477539063e-08	0.7900352306879229

7 метод

k	x	y	beta	Discrapancy
0	-0.166666777922715	1.500001000075189	1	1.219693917604783
1	-0.1267360686994656	2.271414198706363	1	0.7423064572749474
2	-0.03573183044415063	1.885311173579473	1	0.2527830674835531
3	0.006775285345015522	1.816827646083639	1	0.1866615194700237
4	0.01700502617363706	1.982994973840875	1	0.09619843160810292
5	0.01660534129662902	1.998619292025088	0.03125	0.09516059656352278

6	0.008386807502269328	2.071948071205787	0.5	0.09359265554825308
7	0.00452362489990729	1.995476375111322	1	0.0255895521933569
8	0.004510362100646182	1.997429704452277	0.00390625	0.02558814240498664
9	0.003959698142788333	2.007338275656874	0.125	0.0250876180795303
10	-4.567346378233673e-05	2.017209709520621	1	0.01717108990478973
11	-0.0002001009068232314	2.000200100909153	1	0.001131941670756195
12	-0.0002000997617996461	2.000196295704836	7.62939453125e-06	0.001131941585378461
13	-0.0002000526482696569	2.000178622119971	0.000244140625	0.00113187157594908 7
14	-0.0001985000582865453	2.000063539675704	0.0078125	0.001130967306827658
15	-0.0001488969491056916	1.999447992705693	0.25	0.001095772156350175
16	-7.444854995112189e-05	2.000074448550009	1	0.0004211445964517727
17	-7.444849670069387e-05	2.000073974893621	9.5367431640625e-07	0.0004211445615163553
18	-7.444631438137641e-05	2.000071566790711	3.0517578125e-05	0.0004211417945101926
19	-7.437414154713156e-05	2.000056897630078	0.0009765625	0.0004210865022714647
20	-7.205228728311191e-05	1.999976044027969	0.03125	0.000418744090359047
21	-3.60240336900739e-05	1.999747680593703	0.5	0.0003530854569779205
22	-1.80120286699988e-05	2.000018012028694	1	0.0001018914209268714
23	-1.801202705960196e-05	2.000017954033798	5.960464477539063e-08	0.0001018914283223578

- Тест 3 (3 прямых)



  - Начальное приближение снаружи (2, 3)

2 метод

k	x	y	beta	Discrapancy
0	1	1.5	0.5	2.345207879911715
1	1.375	1.3125	0.125	2.309964826572041
2	1.353515625	1.3544921875	0.015625	2.309956156657245
3	1.332366943359375	1.333328247070313	0.015625	2.309401483292774
4	1.333669498562813	1.332677207887173	0.00048828125	2.309401216596098
5	1.333018292752968	1.333979611594259	0.00048828125	2.309401212431026
6	1.332692848833839	1.333653932978147	0.000244140625	2.309401209981536
7	1.333344046868791	1.333328333873416	0.000244140625	2.309401076795834
8	1.333338960564902	1.333323247629467	3.814697265625e-06	2.309401076791686
9	1.333333874280415	1.333333420193982	3.814697265625e-06	2.309401076758653
10	1.333333556388847	1.333333102302523	2.384185791015625e-07	2.309401076758526
11	1.333333397443101	1.333333420193989	1.192092895507813e-07	2.30940107675851
12	1.333333238497374	1.333333261248259	5.960464477539063e-08	2.309401076758512

3 метод

Не находит решение.

7 метод

Не находит решение.

- Начальное приближение внутри (1, 1)

2 метод

k	x	y	beta	Discrapancy
0	1.75	0.75	0.25	2.423839928708165
1	1.3125	1.5625	0.25	2.330168985288406
2	1.1484375	1.3671875	0.125	2.321955962585746
3	1.5048828125	1.1962890625	0.125	2.320072046016256
4	1.41082763671875	1.37152099609375	0.0625	2.313909962653904
5	1.322650909423828	1.285800933837891	0.0625	2.310648338060073
6	1.281318068504333	1.370619654655457	0.03125	2.310334636506826
7	1.366276878863573	1.327787790447474	0.03125	2.309805189647005
8	1.34492880263133	1.307041106221732	0.015625	2.30962660708139
9	1.323914290090215	1.349118588937017	0.015625	2.309483005926487
10	1.33436774989455	1.343848594448982	0.00390625	2.309454127953557
11	1.323943001848499	1.333349777304849	0.0078125	2.309439192032612
12	1.334396349497529	1.328141379737252	0.00390625	2.309410848645853
13	1.331790106627416	1.333359853604953	0.001953125	2.309402090581831
14	1.334395780351413	1.332057744372917	0.0009765625	2.309401683268566
15	1.333744219911788	1.333360450552422	0.00048828125	2.30940115500616
16	1.333418598764349	1.333034923098674	0.000244140625	2.309401107448185
17	1.333255827939304	1.333360480359038	0.0001220703125	2.309401078767685
18	1.333337210512915	1.33331978942641	3.0517578125e-05	2.309401076821704
19	1.333327037957292	1.333340134581825	7.62939453125e-06	2.309401076777154
20	1.333337210507349	1.333335048292859	3.814697265625e-06	2.309401076769165
21	1.333332124229538	1.333329962023297	3.814697265625e-06	2.309401076765823
22	1.333329581100333	1.333335048292748	1.9073486328125e-06	2.309401076763086
23	1.333334667370511	1.333332505157966	1.9073486328125e-06	2.309401076759092
24	1.333334031586997	1.333333776724116	4.76837158203125e-07	2.309401076758933
25	1.333333395803787	1.333333140941027	4.76837158203125e-07	2.309401076758516
26	1.33333323685806	1.333333458832489	1.192092895507813e-07	2.309401076758509
27	1.33333355474951	1.333333299886755	5.960464477539063e-08	2.309401076758522

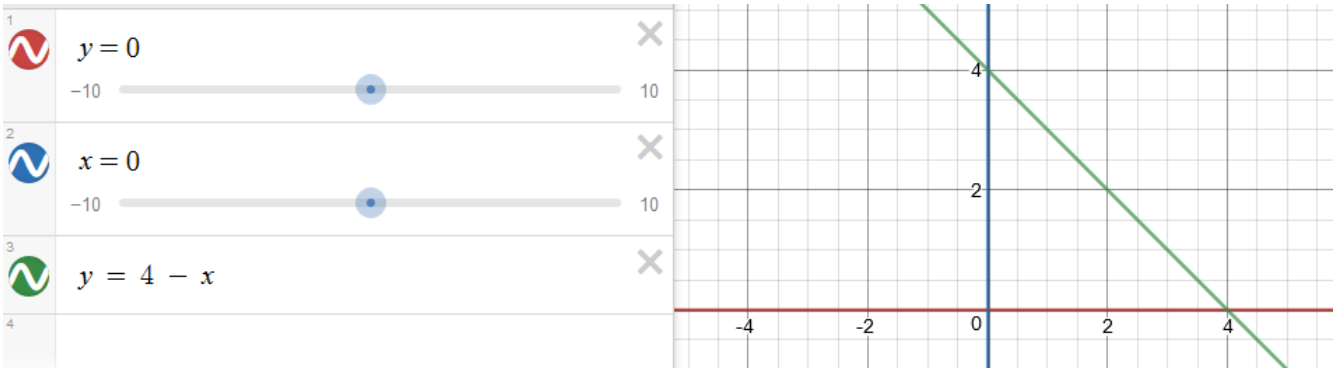
3 метод

Не находит решение.

7 метод

Не находит решение.

- **Тест 4 (3 прямых, 1 взвешена)**



Умножим 3 уравнение на 100

- Начальное приближение снаружи (2, 3)

2 метод

k	x	y	beta	Discrapancy
0	4	0	1	4
1	2	2	0.5	2.82842712474619
2	1.9998779296875	1.9998779296875	6.103515625e-05	2.828359862844556
3	1.999877691283473	1.999877691283473	5.960464477539063e-08	2.828359937679815

3 метод
Не находит решение.

7 метод
Не находит решение.

- Начальное приближение внутри (1, 1)

2 метод

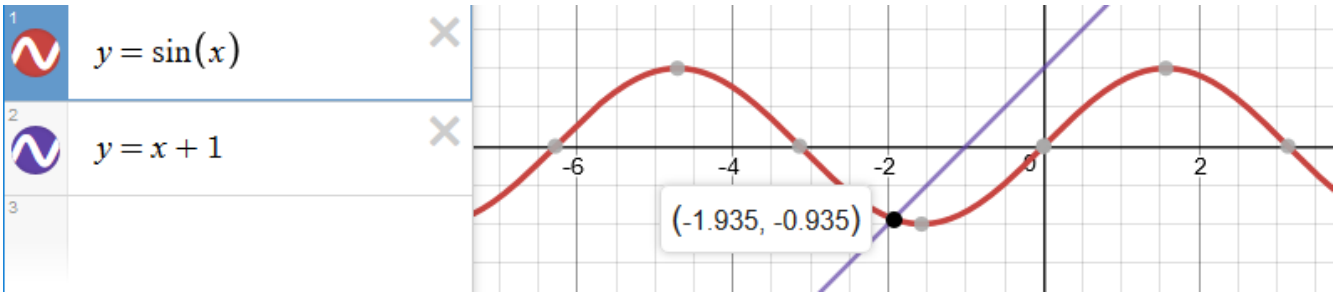
k	x	y	beta	Discrapancy
0	4	0	1	4
1	2	2	0.5	2.82842712474619
2	1.9998779296875	1.9998779296875	6.103515625e-05	2.828359862844556
3	1.999877691283473	1.999877691283473	5.960464477539063e-08	2.828359937679815

3 метод
Не находит решение.

7 метод
Не находит решение.

Вывод: как и ожидалось, решение сместилось из точки (1.33, 1.33) в сторону взвешенной прямой **y = 4 – x**, которую мы умножили на 100, т.е. (1.99, 1.99)

- **Тест 5 (синусоида и прямая)**



- Начальное приближение (3, 5) – точка выше прямой и близка к ней

Все 3 метода:

k	x	y	beta	Discrapancy
0	1.060857014017714	2.060857014017714	1	1.188082881857757
1	-1.260180767263379	-0.2601807672633787	1	0.691964840456824
2	-2.256738202255726	-1.256738202255726	1	0.4829153977648772

3	-1.961088168008833	-0.9610881680088335	1	0.03629010218639017
4	-1.934799723427979	-0.9347997234279789	1	0.0003206893478536754
5	-1.934563230027093	-0.9345632300270934	1	2.613308380805535e-08

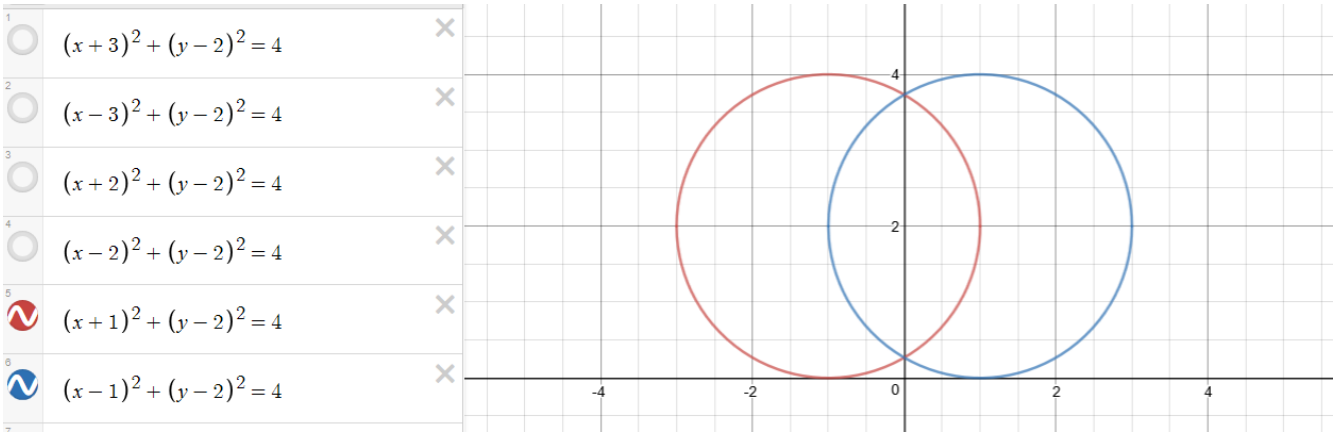
- Начальное приближение (1, 1) – точка между синусоидой и прямой. Близка к синусоиде.

Все 3 метода:

k	x	y	beta	Discrapancy
0	-1.520197577627591	-0.5201975776275911	1	0.4785225787569186
1	-2.024211763070033	-1.024211763070033	1	0.125255506837588
2	-1.937110121074603	-0.9371101210746033	1	0.003456123890821061
3	-1.934565441077953	-0.9345654410779533	1	3.023871841145 72e-06
4	-1.934563210753739	-0.9345632107537386	1	2.324362924355228e-12

Вывод: все 3 метода оказались устойчивы не только на 2 приведённых выше начальных приближениях, но и на других.

- **Тест 6 (влияние размера шага на метод Ньютона при численном вычислении производных)**



Зададим начальное приближение, не лежащее на оси симметрии (4,4)

- h = 1

k	x	y	beta	Discrapancy
0	2.71875	5.568750000000001	0.25	25.41339562821067
1	0.6287451171874991	6.34931861889161	0.5	23.13694581077301
2	-1.881800329581164	5.639862206188434	1	20.21496618528324
3	1.253271087661048	5.323530210161867	1	14.05422524766299
4	-0.8887499127129486	5.280643838754217	1	12.35352328621886
5	0.8468750723346721	4.552024274001724	1	6.443896872139129
6	-0.3170343923208165	4.476553837262294	1	4.660407245682419
7	0.3249701162663987	4.003050059664654	1	1.828624042207163
8	-0.05848436068199647	3.929505899742394	1	1.040536557821693
9	0.07618058952763662	3.786891382573624	1	0.354200860969036
10	-0.005802889535172495	3.766622834102793	1	0.1718909700518917
11	0.01367301655374975	3.741719319120074	1	0.06145616934990708
12	-0.0008033881087843326	3.737861993024655	1	0.02860793695237292
13	0.002319772001814099	3.733696510484228	1	0.01040362702700158
14	-0.0001336784754625312	3.733030175224781	1	0.004814154581257676
15	0.0003907811448305959	3.732327909026797	1	0.001750830014186678

16	-2.232185167450323e-05	3.732215455221187	1	0.000809111544499905
17	6.571751275741008e-05	3.732097414596415	1	0.0002944277233206705
18	-3.752868107723091e-06	3.73207849232136	1	0.0001360428757942121
19	1.104973001319902e-05	3.73205864386988	1	4.950328184983898e-05
20	-6.308083044831023e-07	3.732055462162839	1	2.287248685167804e-05
21	1.857798989532462e-06	3.732052125100891	1	8.323028652996611e-06
22	-1.060589982531839e-07	3.732051590147729	1	3.845556744304176e-06
23	3.123519114440025e-07	3.732051029085365	1	1.399349664251842e-06

- h = 0.01

k	x	y	beta	Discrapancy
0	2.718749999999972	5.639728802992556	0.25	26.10397466905436
1	0.5967670423853457	6.224834670460138	0.5	21.56976878668349
2	-1.751478101353427	5.17559630601601	1	15.18786235867541
3	0.8311414044494714	5.259098288272829	1	11.98842228404905
4	-0.8312793597392338	4.666297312253068	1	7.184107758598095
5	0.392201001472241	4.293090709578132	1	3.587045693258591
6	-0.2034605781206101	3.959776904040011	1	1.373843919155423
7	0.05940007052706969	3.792076788709585	1	0.3474695371874699
8	-0.01203343049451744	3.741494621003712	1	0.05770278131105079
9	0.001110182180661822	3.732750516002387	1	0.004650477784182234
10	-2.565235621122559e-05	3.732071362647242	1	0.0001241165771242258
11	2.487656036931154e-06	3.732052218181885	1	9.862226811959882e-06
12	1.109963719808971e-08	3.732050812370271	1	3.922877361792789e-08

- h = 0.001

k	x	y	beta	Discrapancy
0	2.718749999997014	5.640616015852684	0.25	26.1127041734457
1	0.596363341421434	6.223458512899509	0.5	21.55260885654758
2	-1.750065545296944	5.170514385756937	1	15.13683819223231
3	0.8268449740722765	5.258521745420242	1	11.97095320292371
4	-0.830993328544078	4.660417130352867	1	7.141364668238687
5	0.3882978203044887	4.291036530769327	1	3.566873736776297
6	-0.2028784927455394	3.95610755464854	1	1.354417601128553
7	0.05771993283511806	3.790895143915377	1	0.3396890088462008
8	-0.01189964268449925	3.74097907418522	1	0.05543847358869045
9	0.0009188066097606633	3.732648272198737	1	0.003915447270361688
10	-2.915826741334629e-05	3.732068042487942	1	0.0001180293560693279
11	1.735154238525986e-07	3.732050907729062	1	6.939959491584519e-07

Вывод: При уменьшении размера шага при численном вычислении матрицы Якоби методд Ньютона сходится быстрее.

4. Текст программы

Для удобства программа была разбита на следующие модули:

head.h – заголовочный файл, в котором определяется точность вычислений

SNE.h и SNE.cpp – класс СНУ

main.cpp – файл с исследованиями

head.h

```cpp
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
#include <chrono>
#include <stdio.h>
#include <iostream>
#include <fstream>

using namespace std;


// float || double
typedef double real;
typedef std::vector<real> vec;
typedef std::vector<vector<real>> vec2;

// Умножение на константу
inline bool operator==(const vec& a, const vec& b) {
#ifdef _DEBUG
	if (a.size() != b.size())
		throw std::exception();
#endif
	for (int i = 0; i < a.size(); ++i)
		if (a[i] != b[i])
			return false;

	return true;
}


// Сложение векторов
inline vec operator+(const vec& a, const vec& b) {
#ifdef _DEBUG
	if (a.size() != b.size())
		throw std::exception();
#endif
	vec result = a;
	for (int i = 0; i < b.size(); i++)
		result[i] += b[i];
	return result;
}


// Вычитание векторов
inline vec operator-(const vec& a, const vec& b) {
#ifdef _DEBUG
	if (a.size() != b.size())
		throw std::exception();
#endif
	vec result = a;
	for (int i = 0; i < b.size(); i++)
		result[i] -= b[i];
	return result;
}



// Умножение на константу
inline vec operator*(const vec& a, double b) {
	vec result = a;
	for (int i = 0; i < result.size(); i++)
		result[i] *= b;
	return result;
}


// Умножение на константу
inline vec operator*(double b, const vec& a) {
```

```
	return operator*(a, b);
}


// Скалярное произведение
inline real operator*(const vec& a, const vec& b) {
#ifdef _DEBUG
	if (a.size() != b.size())
		throw std::exception();
#endif
	real sum = 0;
	for (int i = 0; i < a.size(); i++)
		sum += a[i] * b[i];
	return sum;
}


// Потоковый вывод
inline std::ostream& operator<<(std::ostream& fout, const vec& v) {
	for (int i = 0; i < v.size() - 1; ++i)
		fout << v[i] << "\t";
	fout << v.back();
	return fout;
}
```

SNE.h

```
#include "head.h"


class SNE {

public:
	real calcNormE(vec &x);
	void calcVectorF(vec &x, vec &F);
	int inputSNE(int newTestNumber);
	void calcMatrixJAnalitically(vec &x, vec2 &J);
	void calcMatrixJNumerally(vec &x, vec2 &J);
	void deleteRow(int columnToDeleteCount, vec &F_tmp, vec &F, vec2 &Tmp, vec2 &J);
	void svertka(int columnToDeleteCount, vec &F_tmp, vec &F, vec2 &Tmp, vec2 &J);
	void calcGauss(vec &dx, vec &F, vec2 &J);
	void solveSNE();


private:
	int m;					// число функций (число строк)
	int n;					// число переменных (число столбцов)
	int maxiter;			// максимальное число итераций
	int maxiterBeta;		// максимальное число итераций бетты
	real E1;				// Ограничение точности результата
	real E2;				// Ограничение бетты
	real normF_0;			// Норма начального приближения
	int method;				// Номер задания {2,3,7}
	int testNumber;			// Номер теста {1,...7}
	vec x, F;
};
```

SNE.cpp

```
#include "SNE.h"




// Ввод данных
int SNE::inputSNE(int newTestNumber) {

	testNumber = newTestNumber;
	if (testNumber < 1 || testNumber > 7) {
		cout << "This test doesn't exist" << endl;
		return 1;
```

```
        }
        ifstream info("settings.txt"), x_0("x_0.txt");
        info >> m >> n >> maxiter >> maxiterBeta >> E1 >> E2 >> method;
        if (n > m) {
                cout << "Input error" << endl;
                return 2;
        }
        x.resize(n);
        for (int i = 0; i < n; i++)
                x_0 >> x[i];

        calcVectorF(x, F);
        normF_0 = calcNormE(F);
        return 0;
}


// Задание функций
void SNE::calcVectorF(vec &x_tmp, vec &F_tmp) {

        F_tmp.resize(m);
        switch (testNumber)
        {
        case 1: {
                // Тест 1 (окружности не пересекаются)
                F_tmp[0] = pow(x_tmp[0] + 3, 2) + pow(x_tmp[1] - 2, 2) - 4;
                F_tmp[1] = pow(x_tmp[0] - 3, 2) + pow(x_tmp[1] - 2, 2) - 4;
                break;
        }

        case 2: {
                // Тест 2 (одна точка пересечения)
                F_tmp[0] = pow(x_tmp[0] + 2, 2) + pow(x_tmp[1] - 2, 2) - 4;
                F_tmp[1] = pow(x_tmp[0] - 2, 2) + pow(x_tmp[1] - 2, 2) - 4;;
                break;
        }

        case 3: {
                // Тест 3 (2 точки пересения)
                F_tmp[0] = pow(x_tmp[0] + 1, 2) + pow(x_tmp[1] - 2, 2) - 4;
                F_tmp[1] = pow(x_tmp[0] - 1, 2) + pow(x_tmp[1] - 2, 2) - 4;
                break;
        }

        case 4: {
                // Тест 4 (2 окружности и прямая)
                F_tmp[0] = pow(x_tmp[0] + 2, 2) + pow(x_tmp[1] - 2, 2) - 4;
                F_tmp[1] = pow(x_tmp[0] - 2, 2) + pow(x_tmp[1] - 2, 2) - 4;
                F_tmp[2] = x_tmp[0] + x_tmp[1] - 2;
                break;
        }

        case 5: {
                // Тест 5 (три попарно персекающиеся прямые)
                F_tmp[0] = x_tmp[0];
                F_tmp[1] = x_tmp[1];
                F_tmp[2] = x_tmp[0] + x_tmp[1] - 4;
                break;
        }

        case 6: {
                // Тест 6 (три попарно персекающиеся прямые, одна взвешена)
                F_tmp[0] = x_tmp[0];
                F_tmp[1] = x_tmp[1];
                F_tmp[2] = 100 * (x_tmp[0] + x_tmp[1] - 4);
                break;
        }

        case 7: {
                // Тест 7 (синусоида и прямая)
                F_tmp[0] = sin(x_tmp[0]) - x_tmp[1];
                F_tmp[1] = x_tmp[0] - x_tmp[1] + 1;
```

```
                break;
        }
        }
}


// Задание матрицы Якоби аналитически
void SNE::calcMatrixJAnalitically(vec &x, vec2 &J) {

        switch (testNumber)
        {
        case 1: {
                // Тест 1 (окружности не пересекаются)
                J[0][0] = 2 * (x[0] + 3); J[0][1] = 2 * (x[1] - 2);
                J[1][0] = 2 * (x[0] - 3); J[1][1] = 2 * (x[1] - 2);
                break;
        }

        case 2: {
                // Тест 2 (окружности пересекаются в одной точке)
                J[0][0] = 2 * (x[0] + 2); J[0][1] = 2 * (x[1] - 2);
                J[1][0] = 2 * (x[0] - 2); J[1][1] = 2 * (x[1] - 2);
                break;
        }

        case 3: {
                // Тест 3 (окружности пересекаются в двух точках)
                J[0][0] = 2 * (x[0] + 1); J[0][1] = 2 * (x[1] - 2);
                J[1][0] = 2 * (x[0] - 1); J[1][1] = 2 * (x[1] - 2);
                break;
        }

        case 4: {
                // Тест 4 (2 окружности и прямая)
                J[0][0] = 2 * (x[0] + 2); J[0][1] = 2 * (x[1] - 2);
                J[1][0] = 2 * (x[0] - 2); J[1][1] = 2 * (x[1] - 2);
                J[2][0] = 1; J[2][0] = 1;
                break;
        }

        case 5: {
                // Тест 5 (три попарно персекающиеся прямые)
                J[0][0] = 1; J[0][1] = 0;
                J[1][0] = 0; J[1][1] = 1;
                J[2][0] = 1; J[2][1] = 1;
                break;
        }

        case 6: {
                // Тест 6 (три попарно персекающиеся прямые, одна взвешена)
                J[0][0] = 1; J[0][1] = 0;
                J[1][0] = 0; J[1][1] = 1;
                J[2][0] = 100; J[2][1] = 100;
                break;
        }

        case 7: {
                // Тест 7 (синусоида и прямая)
                J[0][0] = cos(x[0]);  J[0][1] = -1;
                J[1][0] = 1; J[1][1] = -1;
                break;
        }
        }
}


// Вычисление матрицы Якоби численно
void SNE::calcMatrixJNumerally(vec &x, vec2 &J) {

        real h = 1e-6;
        vec tmp, l1, l2;
        tmp.resize(n);
```

```
        l1.resize(m);
        l2.resize(m);
        for (int i = 0; i < n; i++)
                tmp[i] = x[i];

        calcVectorF(x, l1);
        for (int i = 0; i < m; i++) {
                for (int j = 0; j < n; j++) {
                        tmp[j] += h;
                        calcVectorF(tmp, l2);
                        for (int k = 0; k < m; k++)
                                l2[k] -= l1[k];
                        J[i][j] = l2[i] / h;
                        tmp[j] = x[j];
                }
        }
}


// Удаление ряда
void SNE::deleteRow(int columnToDeleteCount, vec &F_tmp, vec &F, vec2 &Tmp, vec2 &J) {
        int str_num; // номер убираемой строки
        real min;

        for (int j = 0; j < columnToDeleteCount; j++) {
                min = fabs(F_tmp[0]);
                str_num = 0;
                for (int i = 1; i < m - j; i++) {
                        if (fabs(F_tmp[i]) < min) {
                                min = fabs(F_tmp[i]);
                                str_num = i;
                        }
                }
                swap(Tmp[str_num], Tmp[m - j - 1]);  //меняем строки матрицы
                swap(F_tmp[str_num], F_tmp[m - j - 1]); //меняем функции
        }

        for (int i = 0; i < n; i++) {
                F[i] = F_tmp[i];
                J[i] = Tmp[i];
        }
}


// Свёртка
void SNE::svertka(int columnToDeleteCount, vec &F_tmp, vec &F, vec2 &Tmp, vec2 &J) {

        int str_num; // номер убираемой строки
        real min, sum = 0;
        vec sum1;
        sum1.resize(n, 0);

        for (int j = 0; j < columnToDeleteCount; j++) {
                min = fabs(F_tmp[0]);
                str_num = 0;
                for (int i = 1; i < m - j; i++) {
                        if (fabs(F_tmp[i]) < min) {
                                min = fabs(F_tmp[i]);
                                str_num = i;
                        }
                }

                sum += pow(F_tmp[str_num], 2);
                for (int k = 0; k < n; k++)
                        sum1[k] += F_tmp[str_num] * Tmp[str_num][k];
                swap(Tmp[str_num], Tmp[m - j - 1]);  //меняем строки матрицы
                swap(F_tmp[str_num], F_tmp[m - j - 1]); //меняем функции
        }
        F[m - columnToDeleteCount] = sum;
        for (int k = 0; k < n; k++)
                J[m - columnToDeleteCount][k] = 2 * sum1[k];
```

```
        for (int i = 0; i < n - 1; i++) {
                F[i] = F_tmp[i];
                J[i] = Tmp[i];
        }
}


// Метод Гаусса
void SNE::calcGauss(vec &dx, vec &F, vec2 &J) {

        int maxLine;
        real max, m;
        for (int i = 0; i < n; i++)
                dx[i] = -F[i];

        for (int k = 1; k < n; k++)
        {
                max = J[k - 1][k - 1];
                maxLine = k - 1;
                for (int l = k; l < n; l++)
                        if (J[l][k - 1] > max) {
                                max = J[l][k - 1];
                                maxLine = l;
                        }
                swap(J[maxLine], J[k - 1]);  //поменяли строки в матрице
                swap(dx[maxLine], dx[k - 1]); //поменяли строки в приращении
                for (int j = k; j < n; j++) {
                        m = J[j][k - 1] / J[k - 1][k - 1];
                        for (int i = 0; i < n; i++)
                                J[j][i] = J[j][i] - m * J[k - 1][i];
                        dx[j] = dx[j] - m * dx[k - 1];
                }
        }

        for (int i = n - 1; i >= 0; i--) {
                for (int j = i + 1; j < n; j++)
                        dx[i] -= J[i][j] * dx[j];
                dx[i] = dx[i] / J[i][i];
        }
}


// Решение системы нелинейных уравнений
void SNE::solveSNE() {

        ofstream fout("result.txt");
        cout.precision(std::numeric_limits<real>::digits10 + 1);
        fout.precision(std::numeric_limits<real>::digits10 + 1);
        fout << "k\tx\ty\tbeta\tDiscrapancy\n";
        cout << "k\tx\ty\tbeta\tDiscrapancy\n";

        int columnToDeleteCount;
        vec F_tmp, x_k, x_k1, dx;
        vec2 J, J_tmp;
        J.resize(m);
        for (int i = 0; i < m; i++)
                J[i].resize(n);
        J_tmp.resize(m);
        for (int i = 0; i < m; i++)
                J_tmp[i].resize(n);
        F_tmp.resize(n);
        x_k = x;
        x_k1.resize(n);
        dx.resize(n);

        vec F_tmp1 = F;

        real normF_k = normF_0, normF = normF_0, normF1 = normF_0, beta;
        real nev = normF_k / normF_0;
        if (m - n != 0) {
                switch (method) {
                case 2: columnToDeleteCount = m - n; calcMatrixJAnalitically(x, J_tmp); break;
```

```
                case 3: columnToDeleteCount = m - n + 1; calcMatrixJAnalitically(x, J_tmp); break;
                case 7: columnToDeleteCount = m - n + 1; calcMatrixJNumerally(x, J_tmp); break;
                }
        }
        else //для первой итерации m=n
        {
                columnToDeleteCount = 0;
                calcMatrixJAnalitically(x, J);
                /*calcMatrixJNumerally(x, J);*/
                for (int i = 0; i < m; i++)
                        F_tmp[i] = F_tmp1[i];
        }


        for (int k = 0; k < maxiter && nev > E2; k++) {
                if (normF_k >= normF1 && k)
                        break;
                if (m - n != 0 && k == 0) //для первой итерации
                {
                        switch (method) {
                        case 2: deleteRow(columnToDeleteCount, F_tmp1, F_tmp, J_tmp, J); break;
                        case 3: svertka(columnToDeleteCount, F_tmp1, F_tmp, J_tmp, J); break;
                        case 7: svertka(columnToDeleteCount, F_tmp1, F_tmp, J_tmp, J); break;
                        }
                }
                else {
                        if (m - n != 0) {
                                switch (method) {
                                case 2:  calcMatrixJAnalitically(x_k, J_tmp); deleteRow(columnToDeleteCount,
F_tmp1, F_tmp, J_tmp, J); break;
                                case 3:  calcMatrixJAnalitically(x_k, J_tmp); svertka(columnToDeleteCount,
F_tmp1, F_tmp, J_tmp, J); break;
                                case 7:  calcMatrixJNumerally(x_k, J_tmp); svertka(columnToDeleteCount,
F_tmp1, F_tmp, J_tmp, J); break;
                                }
                        }
                        else if (k) {
                                calcMatrixJAnalitically(x_k, J);
                                /*calcMatrixJNumerally(x_k, J);*/
                                for (int i = 0; i < m; i++)
                                        F_tmp[i] = F_tmp1[i];
                        }
                }
                calcGauss(dx, F_tmp, J);
                normF = normF_k;
                normF1 = normF_k;
                beta = 1;
                for (int k_b = 0; k_b < maxiterBeta; k_b++) {
                        for (int i = 0; i < n; i++)
                                x_k1[i] = x_k[i] + beta * dx[i];
                        calcVectorF(x_k1, F_tmp1);
                        normF_k = calcNormE(F_tmp1);
                        if (normF_k < normF) break;
                        beta /= 2;
                        if (E1 > beta) break;
                }
                for (int i = 0; i < n; i++)
                        x_k[i] = x_k1[i];


                cout << k << "\t" << x_k << beta << "\t" << normF_k << endl;
                fout << k << "\t" << x_k << beta << "\t" << normF_k << endl;


                nev = normF_k / normF_0;
        }  fout.close();
}


// Вычисление Евклидовой нормы
real SNE::calcNormE(vec &x) {
        return sqrt(x*x);
```

```
}
```

main.cpp

```
#include "SNE.h"

void main() {
	SNE sne;
	sne.inputSNE(7);
	sne.solveSNE();
}
```