Kelas B

LAPORAN TUGAS 3
KELOMPOK B01

Analisis Numerik

Anggota Kelompok:

- Fachrur Rozi – 1506689143
- Faridah Nur Suci Amirahmandani – 1506688960
- Jessica Naraiswari Arwidarasti – 1506688935
- Novianti Aliasih – 1506689111
- Nur Intan Alatas – 1506689093

HALAMAN PERNYATAAN ORISINALITAS

Laporan ini adalah hasil kerja Kelompok B01 dan tidak ada bagian dari laporan yang merupakan hasil salinan (*copy*) dari kelompok atau sumber lain. Seluruh data yang dituliskan dalam laporan ini tanpa rekayasa dan dapat dipertanggungjawabkan kebenarannya. Jika kelompok kami terbukti melakukan kecurangan yang tidak semestinya kami lakukan, kami siap menerima konsekuensi yang telah diatur sebelumnya pada aturan perkuliahan Analisis Numerik Semester Genap.

.

DAFTAR ISI

HALAMAN PERNYATAAN ORISINALITAS 2
DAFTAR ISI 3
BAB 1 5
PENDAHULUAN 5
1.1 Topik: Optimisasi Non Linear 5
Metode Steepest Descent 5
Metode Newton 5
Metode Quasi Newton 6
Poblano 7
BAB 2 ISI 8
2.1 Implementasi 8
Fungsi dan gradient dari bagian a , b dan c 8
Steepest Descent: 10
Steepest Desent 10
steepestDescent.m 10
armijoSearch.m 11
Newton Method : 13
Quasi Newton Method : 16
quasiNewton.m 16
Poblano : 18
mainpoblano.m 18
2.2 Hasil Eksperimen 19
Steepest Descent : 19
Newton : 23
Quasi Newton : 28
Poblano 33
2.3 Analisis 40
2.31 Analisis Teknis 40
Steepest Descent : 40
Newton : 40
Quasi-Newton : 41
Poblano : 41
2.32 Analisis Hasil 41
Steepest Descent : 41

Newton : 41
Quasi Newton : 41
Poblano : 42

BAB 3 PENUTUP 43
3.1 Kesimpulan 43
3.2 Log Diskusi 43
Diskusi 1 43
Diskusi 2 43
3.3 Pembagian Kerja 44

DAFTAR PUSTAKA 45

LAMPIRAN 46

BAB 1
PENDAHULUAN

1.1 Topik: Optimisasi Non Linear

Persoalan optimisasi dalam hal komputasi adalah persoalan mencari penyelesaian optimal dari suatu persoalan yang biasanya memiliki beberapa alternatif penyelesaian. Secara matematis, persoalan ini dapat dikategorikan menjadi dua elemen yaitu fungsi sasaran dan daerah kelaikan . Selanjutnya, jika ditinjau berdasarkan fungsi sasaran, persoalan optimisasi dapat dikelompokkan menjadi optimisasi linear dan non-linear. Sedangkan ditinjau dari daerah kelaikan, jika daerah tersebut mencakup seluruh domain fungsi, maka dapat disebut persoalan optimasi tak berkendala (*unconstrained optimization*). Sebaliknya, daerah yang merupakan subset sejati dari domain fungsi disebut persoalan optimasi berkendala (*constrained optimization*).

Secara umum, persoalan optimisasi non linear tak berkendala berfokusuntuk mencari nilai $x^* \in R^n$ yang akan meminimalkan fungsi sasaran *f*. Untuk mencari solusi x^* dibutuhkan beberapa metode optimisasi yang menggunakan beberapa iterasi hingga menemukan solusi x^*. Beberapa metode optimisasi yang dapat digunakan adalah Metode Steepest Descent, Newton, dan Quasi-Newton.

Metode Steepest Descent

Steepest descent merupakan pembaharuan metode penyelesaian persoalan optimisasi dari metode direct line search. Dimana pada metode steepest descent ditentukan arah p = - grad(f). Karena yang dituju adalah titik minimal dari f maka arah pencarian akan mengarahkan kita pada titik terendah yang berlawanan dengan arah gradien. Pada metode steepest descent arah (a0) dapat dicari menggunakan direct line search ataupun backtracking-armijo line search.

Metode Newton

Metode Newton memiliki 2 permasalahan yaitu jika n sangat besar akan terjadi masalah pada kebutuhan penyimpanan dan kebutuhan turunan kedua (matriks Hessian H) serta inversenya yang membutuhkan *cost* yang sangat mahal. Dalam laporan ini terdapat beberapa metode yang mungkin mengatasi beberapa masalah yang ada pada metode Newton. Metode tersebut adalah Limited-memory BFGS, Nonlinear Conjugate

Gradient, dan Truncated Newton.

Metode Quasi Newton

Metode quasi Newton pada dasarnya termasuk dalam kelas metode *Direct Search*. Perbedaannya terletak pada komputasi $\boldsymbol{\alpha}$ karena tentunya Newton ataupun quasi newton memiliki $\boldsymbol{\alpha}$ = 1, sementara untuk metode direct search $\boldsymbol{\alpha}$ merupakan hasil tebakan awal. Akan tetapi, masalah yang muncul berikutnya yaitu biaya komputasi yang besar untuk mencari dan mengevaluasi *p (direction)*. Adapun ide dasar dari metode quasi newton ini adalah dengan membuat matriks B secara iteratif yang merupakan hasil pendekatan/aproksimasi yang diharapkan dapat konvergen ke invers matriks H. Pemilihan B yang tepat bukan hanya akan mengefisiensikan langkah iterasi, namun juga mempercepat laju konvergensi. Untuk itu terdapat 3 rumusan untuk peremajaan matriks B yang banyak dipakai, yakni *Symmetric Rank-1* (SR1), *Davidon-Fletcher-Powell* (DFP), dan *Broyden-Fletcher-Golfarb-Shanno* (BFGS).

- **Metode SR-1**

$$B^{k+1} = B^k + \frac{(\delta^k - B^k\gamma^k)(\delta^k - B^k\gamma^k)^T}{(\delta^k - B^k\gamma^k)^T\gamma^k}$$

- **Metode DFP**

$$B^{k+1} = B^k + \frac{\delta^k(\delta^k)^T}{(\delta^k)^T\gamma^k} - \frac{B^k\gamma^k(\gamma^k)^T B^k}{(\gamma^k)^T B^k\gamma^k}$$

- **Metode BFGS**

$$B^{k+1} = B^k + \left(1 + \frac{(\gamma^k)^T B^k\gamma^k}{(\delta^k)^T\gamma^k}\right)\frac{\delta^k(\delta^k)^T}{(\delta^k)^T\gamma^k} - \left(\frac{\delta^k(\gamma^k)^T B^k + B^k\gamma^k(\delta^k)^T}{(\delta^k)^T\gamma^k}\right)$$

$$\delta^k = x^{k+1} - x^k$$
$$\gamma^k = g^{k+1} - g^k$$
$g = \nabla f$ adalah vektor gradien

Semester Genap 2011/2012

Analisis Numerik

Rumus Peremajaan Matriks B

Sumber : Slide Ajar Bab 5 - Optimisasi Non Linear

Adapun rumusan untuk peremajaan matriks B yang digunakan dalam laporan ini yaitu *Broyden-Fletcher-Golfarb-Shanno* (BFGS) dan komputasi langkah pencarian pada laporan ini menggunakan metode *Direct Line Search*.

Poblano

Terdapat beberapa metode untuk melakukan optimasi non linier tak berkendala, namun pada tugas kali ini ada 2 metode yang digunakan yaitu Limited Memory BFGS dan Nonlinear Conjugate Gradient. Kedua metode tersebut dapat langsung digunakan dengan melakukan instalasi Poblano Toolbox pada Matlab. Limited Memory BFGS (LBFGS) adalah suatu algoritma optimasi yang digunakan pada metode Quasi-Newton, sama seperti BFGS. Namun bedanya terdapat pada memori yang terbatas. Lalu, terdapat juga Nonlinear Conjugate Gradient, yaitu suatu algoritma yang digunakan untuk mengoptimasi fungsi non-linier seperti energy-minimization.

BAB 2 ISI

2.1 Implemententasi

Fungsi dan gradient dari bagian a , b dan c

Fungsi dibawah ini menjalankan steepest descent untuk fungsi a,b dan c

```
% Soal 1 - STEEPEST DESCENT
% =========================
%
% a) f(x, y) = 0.5x^2 + 2.5y^2, dengan tebakan awal (x, y) = (1,2)
  arrtol = [1e-4, 1e-6, 1e-8, 1e-10, 1e-12];
  initial = [1,2]';

  function y = a(x)
    y = 0.5 * x(1).^2 + 2.5 * x(2).^2;
  end;

  function y = grad_a(x)
    y(1) = x(1);
    y(2) = 5 * x(2);
    y = y';
  end;

  a = [];
  tol,iter , time, sol
  for i = 1:length(arrtol)
    tol = arrtol(i)
    tic;
      [sol iter] = steepestDescent(@a, @grad_a, initial, tol);
    elapsed_time = toc;
    a =[a; [tol iter time sol ]];
  end;

  csvwrite("soal1a.csv",a);
% b) Fungsi luas permukaan tabung dengan syarat volumenya adalah 400
satuan isi dengan tebakan awal (r,t, λ) = (1,1, -0.5)


  initial = [1,1,-0.5]';

  b = []
```

```
  function L = b(x)
     f = 2 * pi * x(1) * (x(1) * x(2));
     g = pi * x(1).^2 * x(2) - 400;

     L = f + x(3) * g;
  end;

  function y = grad_b(x)
    y(1) = (4 * pi * x(1)) + (2 * pi * x(2)) + (2 * x(3) * pi * x(1)
* x(2));
    y(2) = (2 * pi * x(1)) + (x(3) * pi * x(1).^2);
    y(3) = (pi * x(1).^2 * x(2)) - 400;
    y = y';
  end;

  b = [];
  for i = 1:length(arrtol)
    tol = arrtol(i)
    tic;
      [sol iter] = steepestDescent(@b, @grad_b, initial, tol);
    elapsed_time = toc;
    b =[b; [tol iter time sol]];
  end;
  csvwrite("soal1b.csv",b);

% c) Griewank function
  t = [1,2,3];
  a = 0; b = 1;
  initial = [-400, -200, 200, 400];

  function y = c(x)
    n = length(x);
    sum = 0;
    mul = 1;

    for i = 1:n
      sum = sum + ((x(i).^2)/4000);
      mul = mul * (cos(x(i)/ sqrt(i)));
    end;

    y = 1 + sum -  mul;
  end;

  function y = grad_c(x)
```

```
    n = length(x);
    for d = 1 : n
      sum = 0;
      mul = 1;
      for i = 1:n
        if(i == d)
          sum = sum + ((x(i))/2000);
          mul = mul * 1/sqrt(i) * (sin(x(i)/ sqrt(i)));
        else
          mul = mul * (cos(x(i)/ sqrt(i)));
        end;
      end;
      y(d) = sum +  mul;
    end;
    y = y';
  end;

  c = [];
  for index = 1 : length(t)
    initial = [initial initial]
   for i = 1:length(arrtol)
    tol = arrtol(i)
    tic;
      [sol iter] = steepestDescent(@c, @grad_c, initial, tol);
    elapsed_time = toc;
    c =[c; [index tol iter time sol]];
  end;
  csvwrite("soal1c.csv",c);
  end;
```

Pada program diatas didefinisikan fungsi a , b dan c beserta turunannya secara manual. Pada program diatas didefinisikan juga generate function dari input untuk griewank function. Program diatas akan menuliskan pada file .csv data-data yang dibutuhkan untuk membuat grafik

Steepest Descent:

Steepest Desent

steepestDescent.m

```
% STEEPEST DESCENT ALGORITHM
```

```
%=============================
% Nur Intan - 1506689093
%
% Input
% [f]          : function               e.g @c
% [grad]       : gradient function      e.g @grd_c
% [initial]    : initial x              e.g [1, 2, 3]
% [tol]        : stoping tolerance      e.g 1e-4
%
% Output
% [x]     : minimum solution of f(x)
% [k]     : minimum step to find minimum solution
function [x k] =  steepestDescent(f, grad, initial, tol)
  x = initial;
  k = 0;
  while(norm(grad(x, f), 2) > tol)
    p = -grad(x, f);
    a = armijoSearchV2(f, grad, x, 1, p)
    x = x + a*p
    k = k + 1
  end;
  x, k
end;
```

Pada kode diatas dilakukan steepest descent dengan algoritma line search backtracking armijo. Kode diatas akan berhenti ketika sudah menemukan solusi (fungsi gradient(x) = 0). program diatas akan mengupdate nilai x dengan menambahkan x dengan a dan p, dimana a merupakan panjang langkah dan p merupakan arah. Steepest descent diatas memerlukan fungsi dan gradient sebagai parameter.
Kompleksitas dari algoritma diatas bergantung dari fungsi dan nilai initial nya

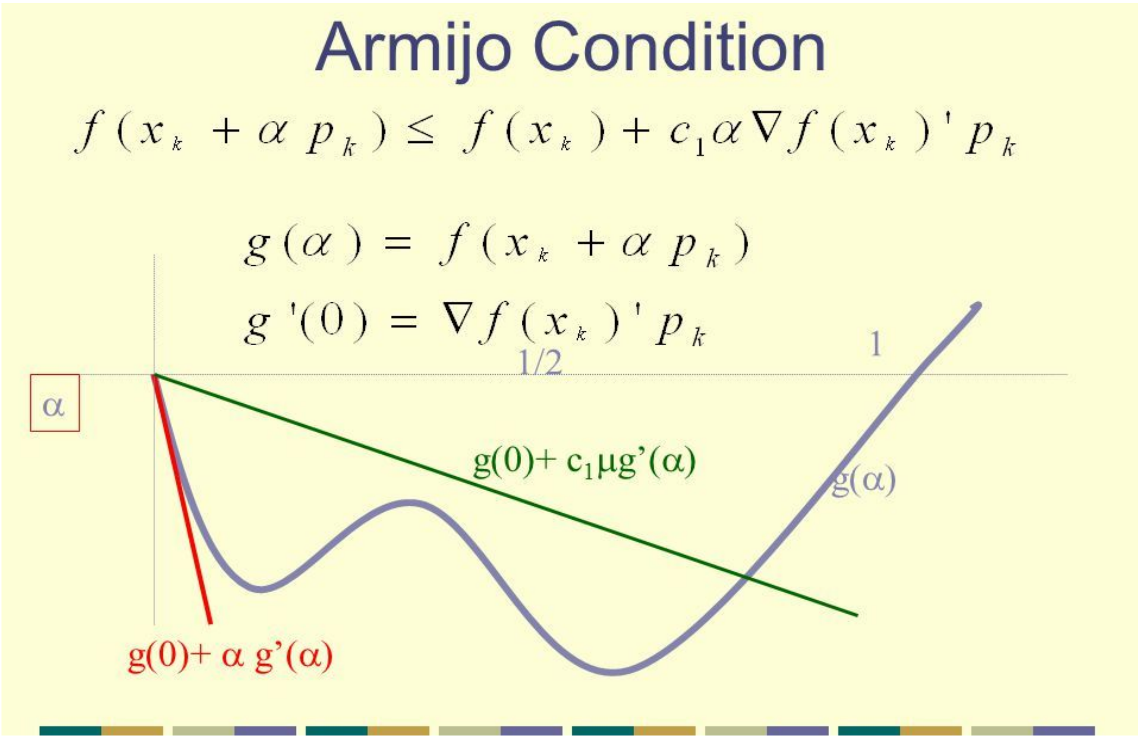
armijoSearch.m

```
% ARMIJO BACKTRACKING ALGORITHM
%=============================
% Nur Intan - 1506689093
%
% Input
% [f]     : function             e.g @c
% [grad]  : gradient function    e.g @grad_c
% [a]     : search step, usually  e.g 1
% [p]     : search direction     e.g 3
%
% Output
% [a]     : final minimum step
function [a] = armijoSearchV2(f, grad, x, a , p)
  BETA = 0.1;
```

```
  TAU =  0.5;

  i = 0;
  while(f(x + a * p) > f(x) + a * BETA * grad(x,f)' * p)
    a = a * TAU;
    i = i + 1;
  end;
end;
```

Kode diatas merupakan implementasi dari armijo backtracking line search algorithm. Kode diatas akan selalu mengecek kondisi armijo dan dapat dipastikan kekonvergenannya, hal ini dapat dibuktikan dengan gambar dibawah ini :



Kompleksitas dari algoritma diatas bergantung dari fungsi dan nilai awal , tau dan beta nya

Newton Method :

fungsi mencari matriks gradien per fungsi

```
## manual memasukkan hasil turunan parsial pertama dari setiap fungsi
function [res] = delta_function_a(x)
    y(1) = x(1);
    y(2) = 5 * x(2);
    res = y';
end
####################################
function [res] =  delta_function_b(x)
    y(1) = (4 * pi * x(1)) + (2 * pi * x(2)) + (2 * x(3) * pi * x(1) *
x(2));
    y(2) = (2 * pi * x(1)) + (x(3) * pi * x(1).^2);
    y(3) = (pi * x(1).^2 * x(2)) - 400;
    res = y';
end
####################################
function [res] = delta_function_c(x)
    n = length(x);
    for d = 1:n
      sum = 0;
      mul = 1;
      for i = 1:n
        if(i == d)
          sum = sum + ((x(i))/2000);
          mul = mul * 1/sqrt(i) * (sin(x(i)/ sqrt(i)));
        else
          mul = mul * (cos(x(i)/ sqrt(i)));
        end
      end
      y(d) = sum +  mul;
    end
    res = y';
end
```

fungsi mencari matriks hessian per fungsi

```
# manual menentukan matriks turunan parsial kedua dari masing-masing fungsi
function [res] = hessian_function_a(x)
    y = zeros(2);
    y(1,1) = 1;
    y(2,2) = 5;
    res = y;
end
#######################################
function [res] = hessian_function_b(x)
```

```
    y = zeros(3);
    y(1,1) = 4*pi + 2*pi*x(2)*x(3);
    y(1,2) = 2*pi + 2*pi*x(1)*x(3);
    y(1,3) = 2*pi*x(1)*x(2);
    y(2,1) = y(1,2);
    y(2,3) = pi*x(1)^2;
    y(3,1) = y(1,3);
    y(3,2) = y(2,3);
    res = y;
end
#######################################
function [res] = hessian_function_c(x)
    n = length(x);
    y = zeros(n);
    for i = 1:n
        changed_x = (1/sqrt(i))*sin(x(i)/sqrt(i));
        for j = 1:n
            if i == j
                mul = 1;
                for k = 1:n
                    if k == i
                        mul =
mul*(1/sqrt(k))*(1/sqrt(k))*cos(x(k)/sqrt(k));
                    else
                        mul = mul*cos(x(k)/sqrt(k));
                    end
                end

                r = 1/2000 + mul;
                y(i,j) = r;
            else
                mul = 1;
                for k = 1:n
                    if k == i
                        mul = mul*changed_x;
                    elseif k == j
                        mul = mul*-(1/sqrt(k))*sin(x(k)/sqrt(k));
                    else
                        mul = mul*cos(x(k)/sqrt(k));
                    end
                end
                y(i,j) = mul;
            end
        end
    end
    res = y;
end
```

implementasi newton

```
# x adalah tabakan awal
# A adalah fungsi matriks hessian
# b adalah fungsi matriks gradien
# tol adalah toleransi

function [res] = newton(x, A, b, tol)
    x_new = x';
    residu = norm(b(x_new));
    iter = 0;
    while residu > tol
        v = A(x_new) \ -b(x_new);
        x_new = x_new + v;
        residu = norm(b(x_new));
        iter = iter + 1;
    end
    res = x_new;
end
```

Kompleksitas Waktu	Kompleksitas Memory
penyelesaian persamaan v = O(N^3) jumlah konvergensi = M kompleksitas = O(M*N^3)	matriks Hessian = N*N matriks grad = 1*N matriks v = 1*N kompleksitas memory = 2N + N^2

Quasi Newton Method :

quasiNewton.m

```
% Reference :
https://github.com/yyc9268/Numerical_optimization/blob/49639aa66b0f1dd47803
110ae47b1083e4217b51/matlab/multivariate_smooth/quasi_newton.m
% with some modification

function minX = quasiNewton(f, initial, tol, max_iter)
  x = initial;
  k = 0;
  n = length(initial);
  B = eye(n);

  while k < max_iter
    g = grad(f,x);

    % Cek sudah konvergen apa belum
    if abs(g) <= tol
      break
    end

    % a) Hitung p^k = -B^kg^k.
    % Return vektor p (arah pencarian)
    p = -B * g;

    % b) Hitung panjang langkah (solusi dari min f(x0 + alpha.p^k))
menggunakan line search.
    % Return alpha
    alpha = directLineSearch(f, x, 1, p);

    % c) Iterasi x^(k+1) = x^k + alphak.p^k
    % Return x^k (tebakan baru)
    x_old = x;
    x = x + a*p;

    % d) Hitung B^(k+1) dengan BFGS
    % Return B^k+1 (matriks peremajaan)
    g_new = grad(f, x);
    deltaX = x - x_old;
    Y = g_new - g;
    B = bfgs(B, deltaX, Y);

    % Iterasi
    k++;
  endwhile;
  minX = x
```

```
end;

% Direct Line Search
% Param f : fungsi
% x : tebakan awal
% p : tebakan arah pencarian
% alpha : tebakan panjang langkah
%
function a = directLineSearch(f, x, p, alpha)
  % Hitung x^1 = x^0 + alpha0.p^0
  x_new = x + alpha.p;

  f_alpha = f(x_new);

  % cek memenuhi kriteria solusi minimal;
  while norm(f_a,2) > 0.00001
    a = a + alpha * direction;
    f_a = f(x + a*p);
  end;
end;

% Broyden-Fletcher-Goldfarb-Shanno
% Param B : matriks identitas ukuran x
% deltaX : selisih tebakan1 dan tebakan0
% Y : selisih grad1 dan grad0
function B = bfgs(B, deltaX, Y)
    N = length(deltaX);

    a = Y' * Y;
    b = deltaX' * Y;
    c = deltaX * deltaX';
    d = deltaX' * Y;
    e = deltaX * Y';

    B_old = B;

    B = B_old + ((1 + (a/b) *B) * (c/b)) - ((d + e)/e) * B;
end

% Find gradient of function
function H = grad(myFx, X)
    N = length(X);
    for iVar=1:N
        xt = X(iVar);
        h = 0.01 * (1 + abs(xt));
        X(iVar) = xt + h;
        fp = feval(myFx, X, N);
        X(iVar) = xt - h;
        fm = feval(myFx, X, N);
```

Kompleksitas Waktu
quasi : - looping: N direct search : - looping: N bfgs : 1 grad : n   Total:n (n + 1 + n) = n(1 +2n) = n + 2n^2

Poblano :

mainpoblano.m

```
function mainpoblano()
  warning off;
  diary('result_soal2.txt');
  diary on;

  fprintf('------ FUNCTION A ------\n');
  poblanoFunction(@AFunction, [1, 2]');

  fprintf('------ FUNCTION B ------\n');
  poblanoFunction(@BFunction, [1, 1, -0.5]');

  fprintf('------ FUNCTION C ------\n');
  x = [-400, -200, 200, 400, -400, -200, 200, 400];
  poblanoFunction(@GRFunction, [x]');
  poblanoFunction(@GRFunction, [x x]');
  poblanoFunction(@GRFunction, [x x x]');
  poblanoFunction(@GRFunction, [x x x x]');
  poblanoFunction(@GRFunction, [x x x x x]');
end;

function poblanoFunction(fx, x)
  for tol  = [10e-4, 10e-6, 10e-8, 10e-10, 10e-12];
    fprintf('------------ Limited Memory BFGS tol = %d ------------\n',
tol);
    tic;
    res = lbfgs(fx, x, 'Display', 'final');
```

```
    toc;
    fprintf('------------ Nonlinear Conjugate Gradient tol = %d
------------\n', tol);
    tic;
    res = ncg(fx, x, 'Display', 'final');
    toc;
  end;
end;
```

Pada kode tersebut dijalankan fungsi lbfgs dan ncg yang terdapat pada Poblano Toolbox. Lalu, fungsi tersebut dipanggil berkelanjutan dengan variasi tol yang berbeda serta fungsi yang akan dioptimasi. Hasil dari program ini dapat dilihat pada Soal_2/result_soal2.txt

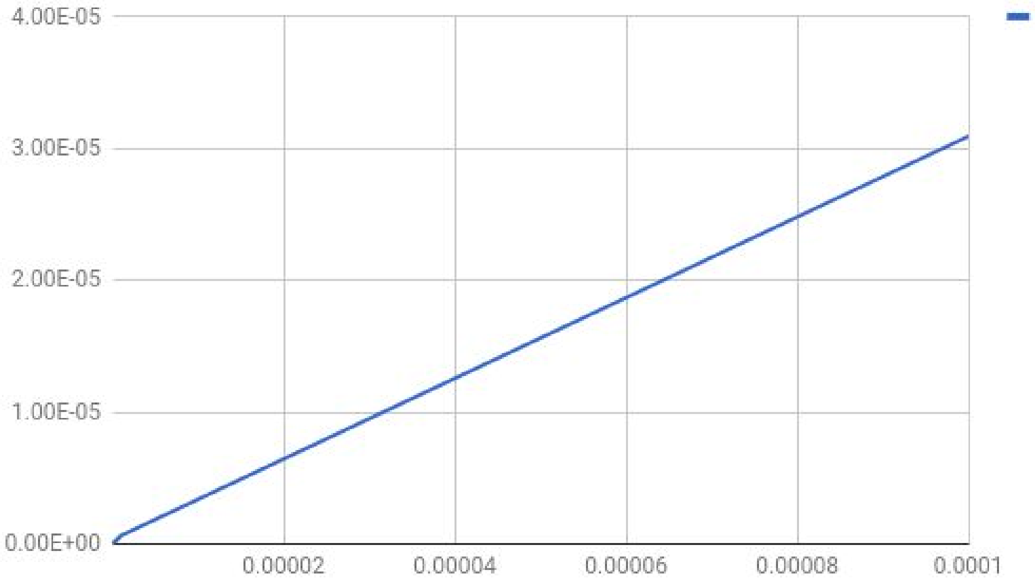
2.2 Hasil Eksperimen

Steepest Descent :

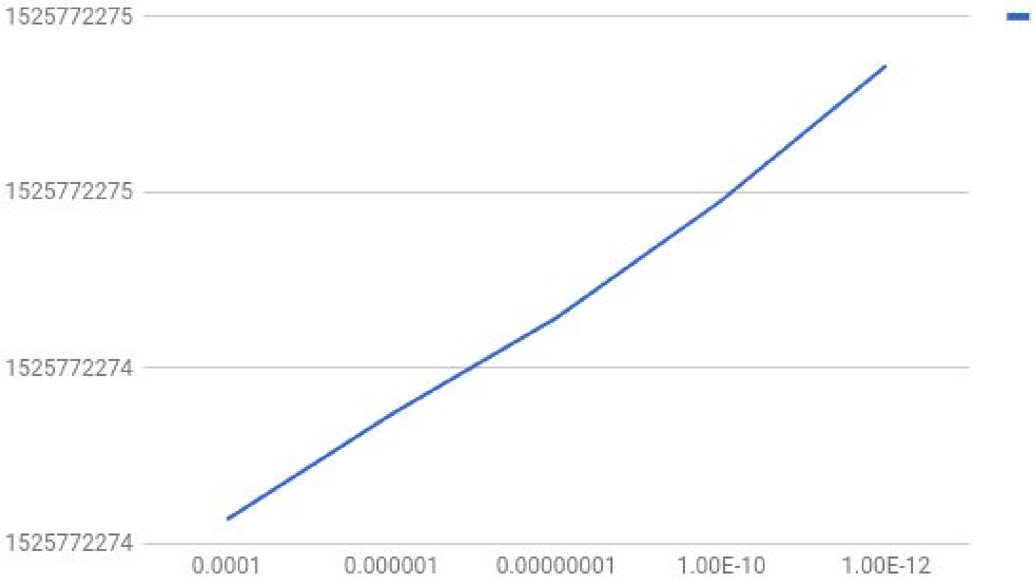
$f(x, y) = 0.5x$^2 + $2.5y$^2 dengan tebakan awal (x, y) = (1,2)

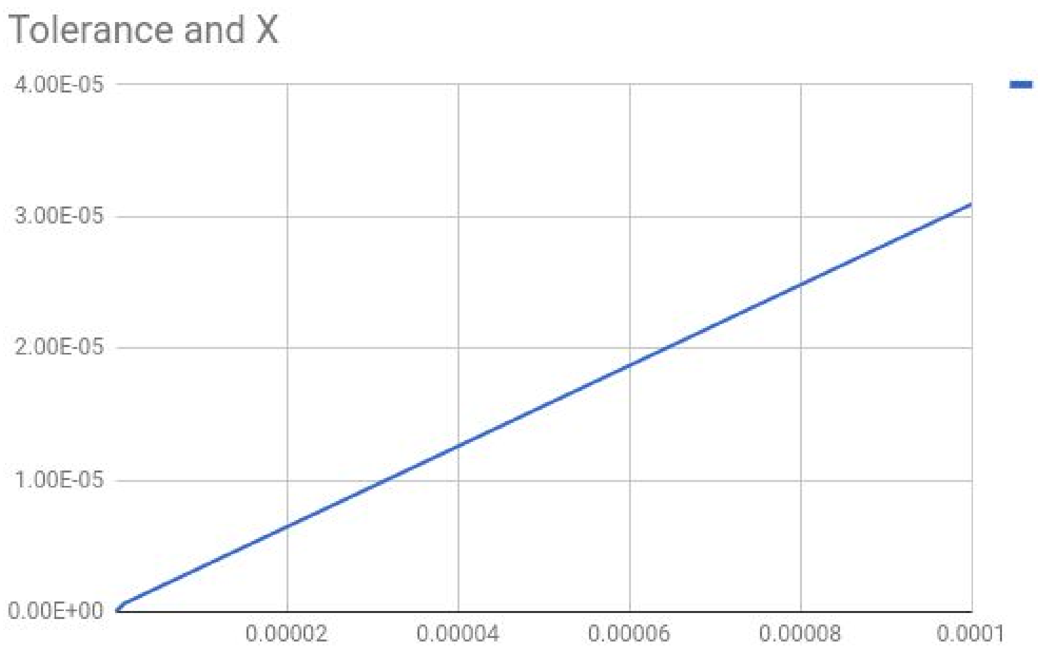
Tolerance	Iteration	time	x	y
0.0001	22	1525772274.46276	3.0933937523514E-05	6.87420833855867E-06
0.000001	30	1525772274.47475	6.11730698096835E-07	1.3594015513263E-07
0.00000001	40	1525772274.48575	4.53645506706332E-09	0.000000001
1E-10	50	1525772274.49906	3.36413141264767E-11	7.47584758366148E-12
1E-12	58	1525772274.5144	6.65270127989407E-13	1.47837806219868E-13

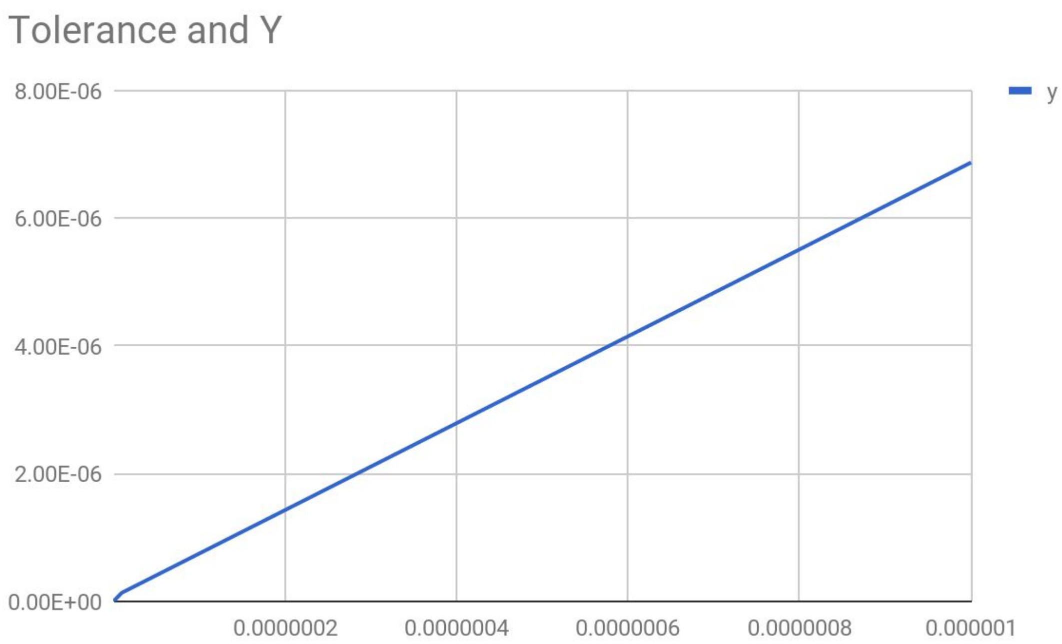
Tolerance and Iteration



Tolerance and Time







Fungsi luas permukaan tabung dengan syarat volumenya adalah 400 satuan isi dengan tebakan awal $(r, t, \lambda) = (1, 1, -0.5)$

Tolerance	Iteration	time	r	t	lamda
0.0001	6	1525774094.11324	-Inf	-Inf	Inf

0.000001	6	1525774094.11571	-Inf	-Inf	Inf
0.00000001	6	1525774094.11799	-Inf	-Inf	Inf
1E-10	6	1525774094.12019	-Inf	-Inf	Inf
1E-12	6	1525774094.12239	-Inf	-Inf	Inf

Griewank function

Griwank function tidak dapat dijalankan pada program dengan t =3, 4, 5 dan toleransi 1e-4 dan 1 e-6 sehingga kami hanya dapat mengumpulkan data seperti dibawah ini :

initial = [-400, -200, 200, 400, -400, -200, 200, 400]

Tolerance	Iteration	time	x1	x2	x3	x4	x5	x6	x7	x8
0.0001	1120	1525788983.9784	-266.8542093323	-133.1047728589	130.3273242551	256.9324965365	-252.0680465956	-130.2920217253	124.0919483259	247.5349016017
0.000001	1198	1525788987.95725	-266.854104 6532	-133.104656 9267	130.3271387617	256.932022429	-252.0674248911	-130.2915175174	124.0913052834	247.533674823

initial = [-400, -200, 200, 400, -400, -200, 200, 400, -400, -200, 200, 400, -400, -200, 200, 400]

tol	iter	time	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16

0.0001	4548	1525789991.54963	-47.0997797995	-22.1916926437	21.7322160831	43.8924660389	-49.0482707094	-23.0152129271	24.8465999633	44.2478311155	-46.9080971811	-19.7679946988	20.7223019611	43.2658400037	-45.0096596705	-23.3421169196	24.1488663791	49.8569978007
0.0000001	4621	1525790050.83573	-47.0997811677	-22.1916940275	21.7322182755	43.892472434	-49.048280462	-23.0152190108	24.8466085386	44.2478508664	-46.9081257791	-19.7680062607	20.7223166826	43.2658981203	-45.0098159583	-23.3423313046	24.1493564511	49.8584678877

Newton :

a. Fungsi A

Tol	Iter	time	x	y
0.0001	1	0.0024	0	0
0.000001	1	4.1828e-04	0	0
0.00000001	1	8.8251e-04	0	0
1E-10	1	0.0013	0	0
1E-12	1	0.0019	0	0

b. Fungsi B

Tol	Iter	time	r	t	lambda
0.0001	4403	1.8404	Nan	Nan	Nan
0.000001	4403	2.7381	Nan	Nan	Nan

0.0000000 1	4403	2.3673	Nan	Nan	Nan
1E-10	4403	2.3538	Nan	Nan	Nan
1E-12	4403	2.3057	Nan	Nan	Nan

tebakan awal 1,1,-05 tidak dapat menemukan titik konvergensi, walaupun nilai tolerance diturunkan hingga 10^-12. tebakan awal yang berhasil mencapai konvergensi, salah satunya adalah [1,1,1]. memerlukan 10 kali iterasi untuk tolerance 10^-4.

c. Fungsi C

i. t = 1

Tol	Iter	time
0.0001	357	0.357
0.000001	358	0.0422
0.0000000 1	358	0.0325
1E-10	358	0.0258
1E-12	359	0.0284

ii. t = 2

Tol	Iter	time
0.0001	1	0.0165
0.000001	1	0.0183
0.0000000 1	1	0.0195
1E-10	1	0.0168

1E-12	1	0.0212

iii. t = 3

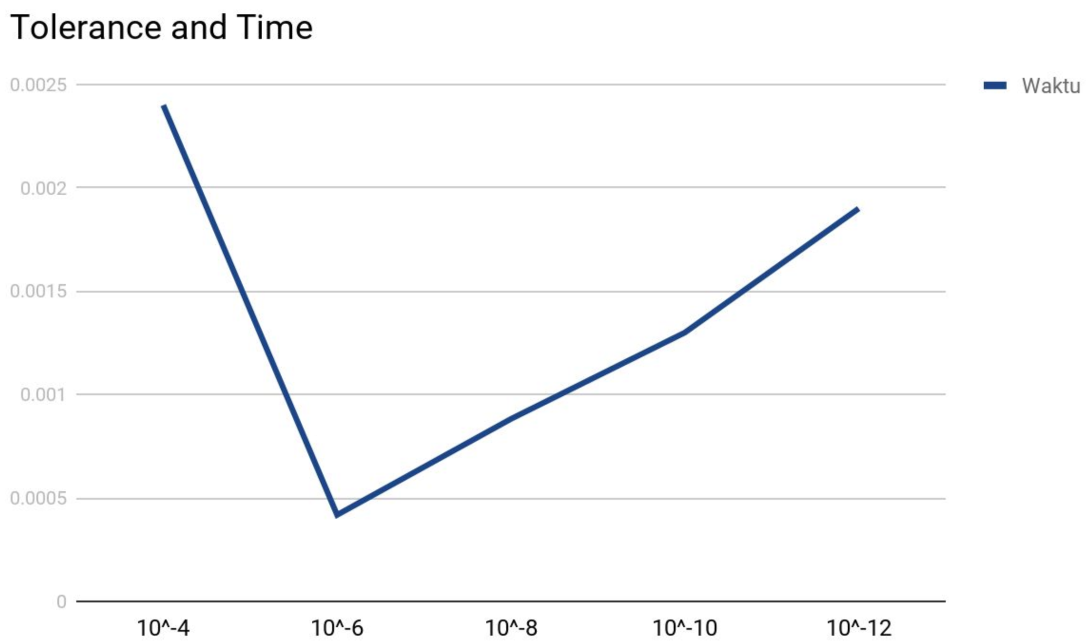
Tol	Iter	time
0.0001	1	7.4819
0.000001	1	7.9128
0.00000001	1	7.1547
1E-10	1	8.0997
1E-12	1	7.2650

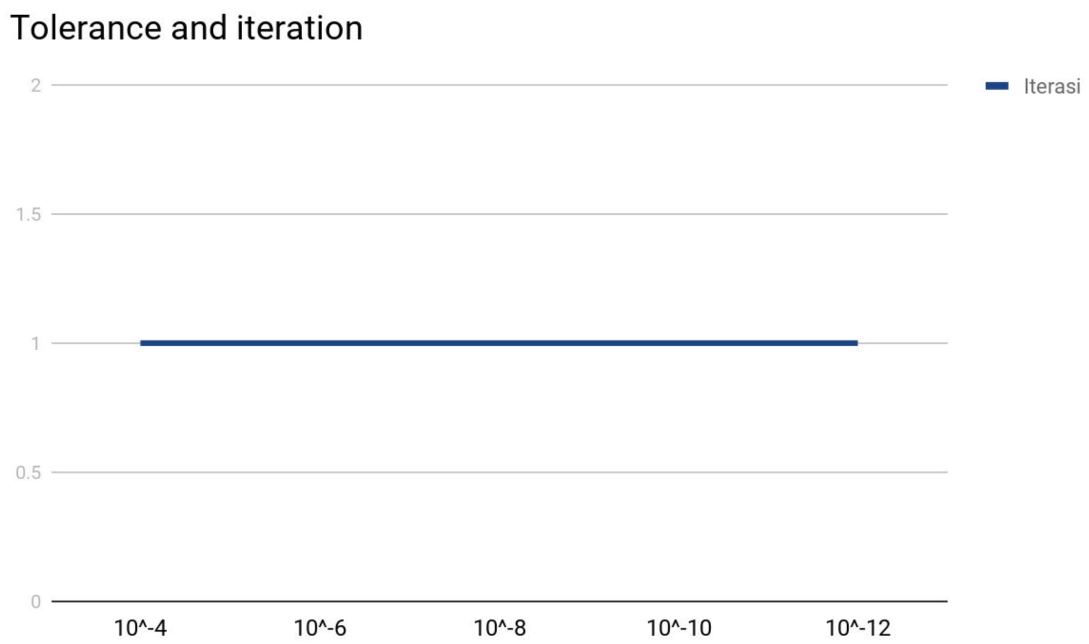
iv. t = 4, t = 5

tidak mendapatkan hasil, karena memory limit karena harus membuat matriks berukuran 4096x4096 dan 32768x32768, lalu dilakukan operasi hessian.

GRAFIK NEWTON

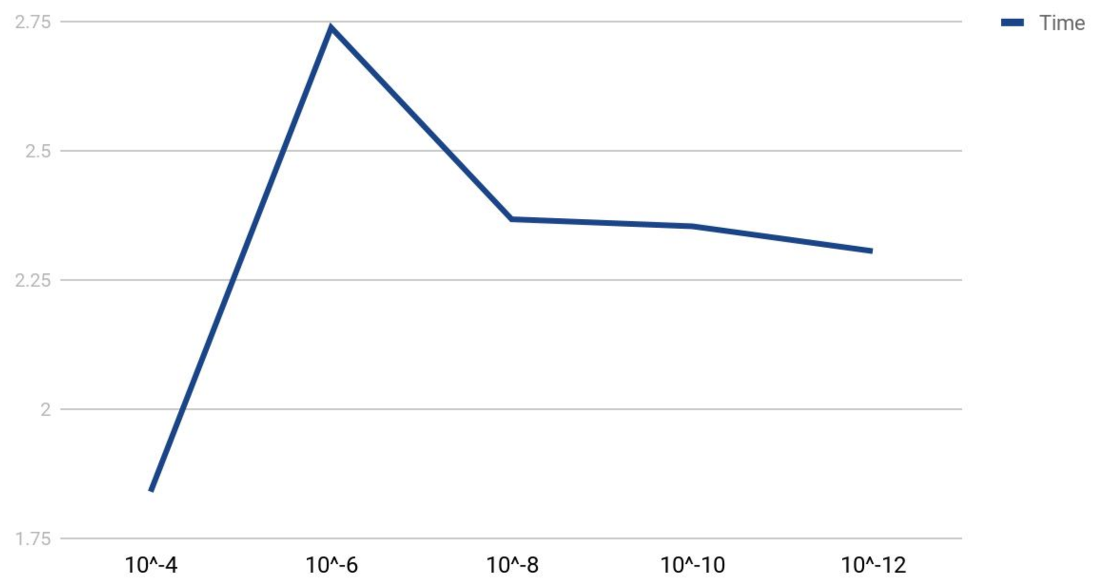
fungsi a


Tolerance and Time
Waktu
0.0025
0.002
0.0015
0.001
0.0005
0
10^-4
10^-6
10^-8
10^-10
10^-12

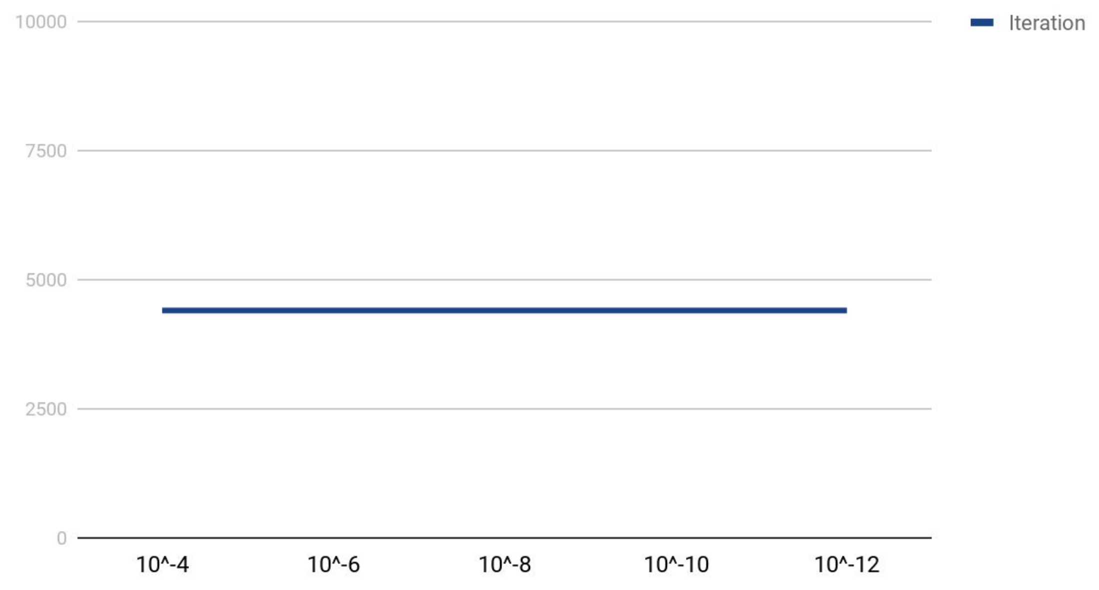

Tolerance and iteration
Iterasi
2
1.5
1
0.5
0
10^-4
10^-6
10^-8
10^-10
10^-12

Fungsi B

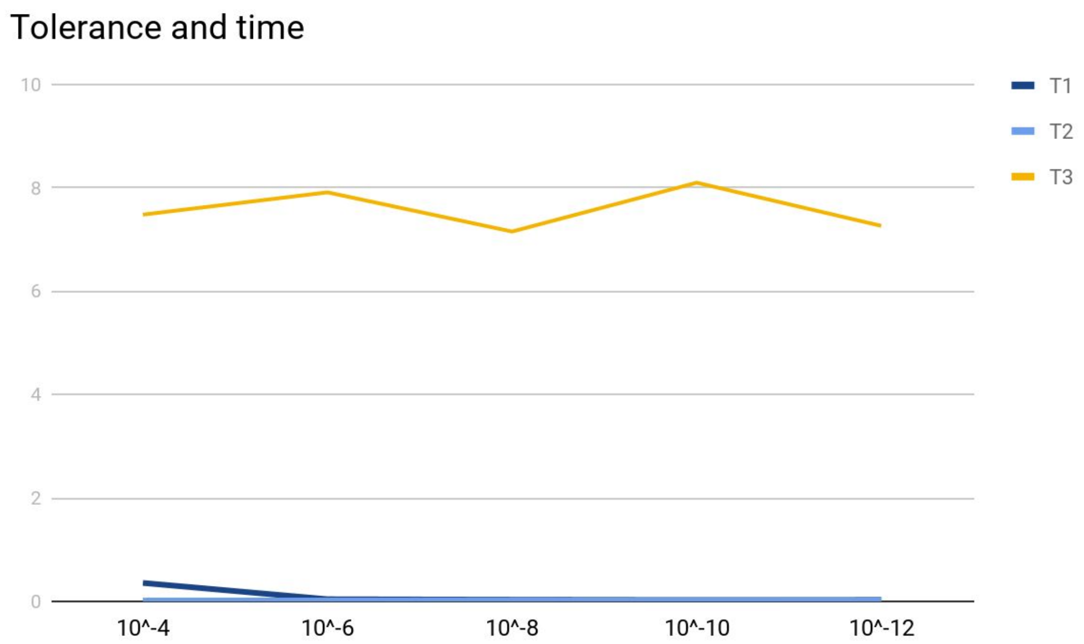
Tolerance and time

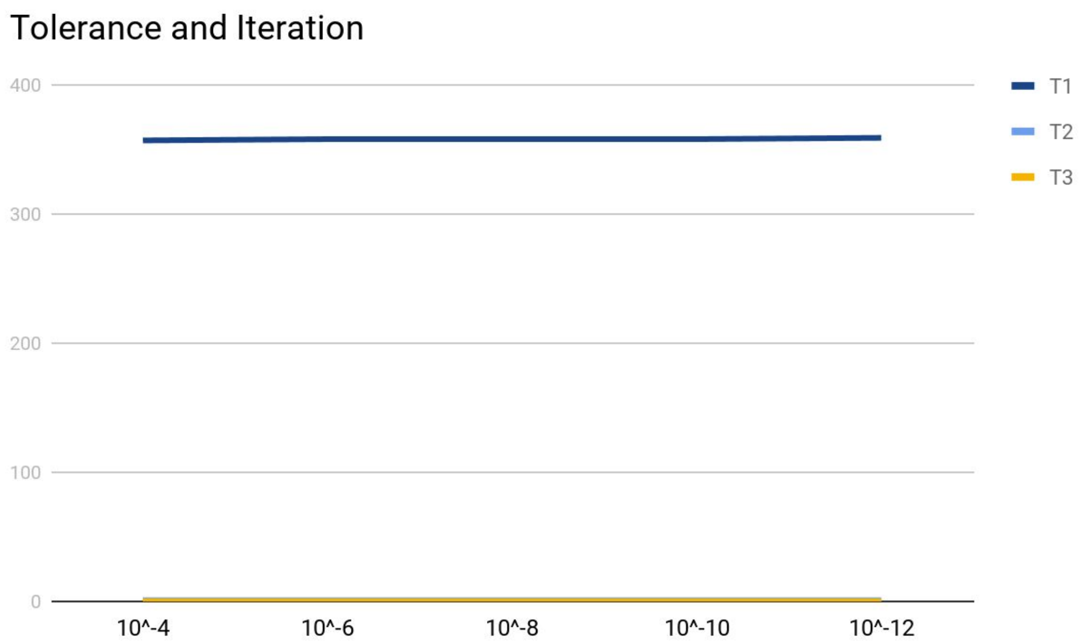


Tolerance and Iteration



Fungsi C untuk t = 1, t=2, t = 3





Quasi Newton :

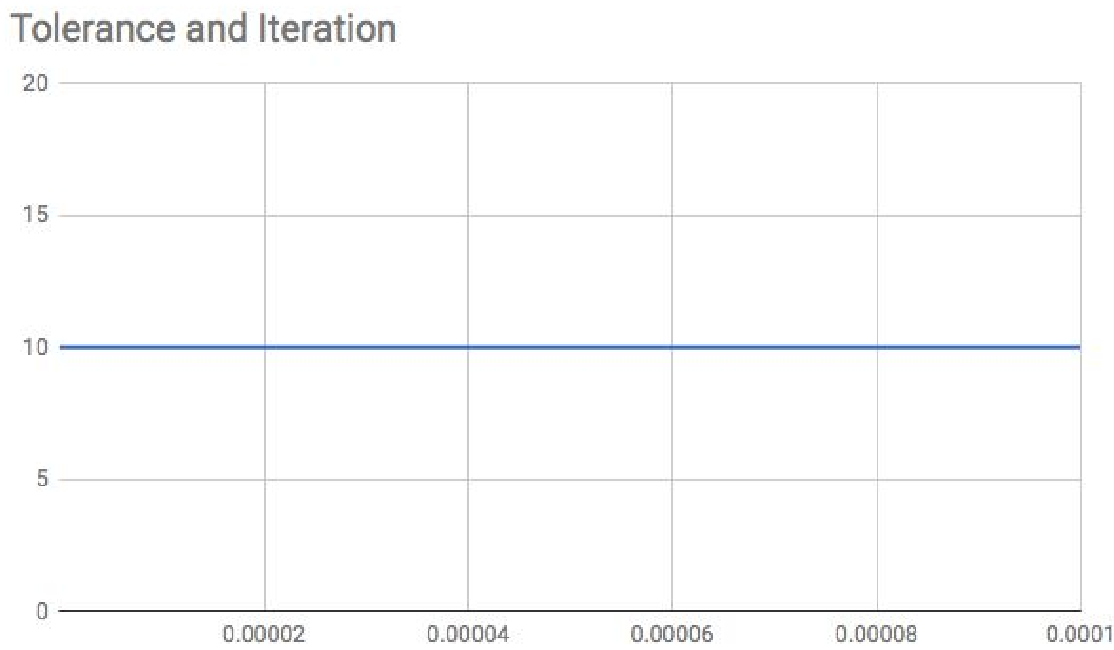
a. Fungsi A

Dari fungsi A, didapatkan hasil eksperimen sebagai berikut.

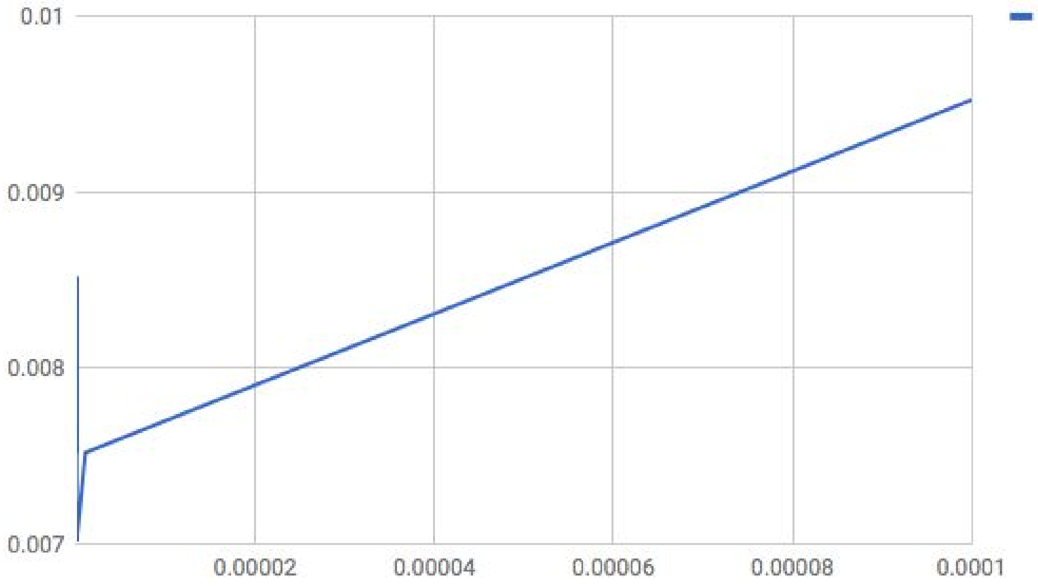
Tol	Iter	time	x	y

0.0001	10	0.0095260	-0.2333739	-0.0090771
0.000001	10	0.0075200	-0.2333739	-0.0090771
0.00000001	10	0.0070190	-0.2333739	-0.0090771
1E-10	10	0.0085220	-0.2333739	-0.0090771
1E-12	10	0.0075209	-0.2333739	-0.0090771

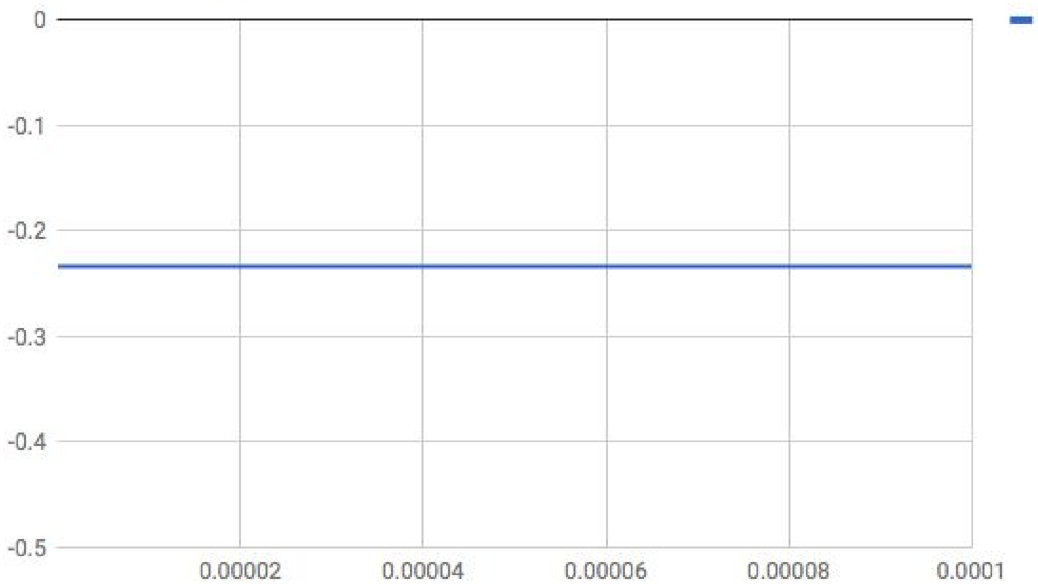
Dari hasil eksperimen tersebut, dapat direpresentasikan kedalam bentuk grafik, yakni

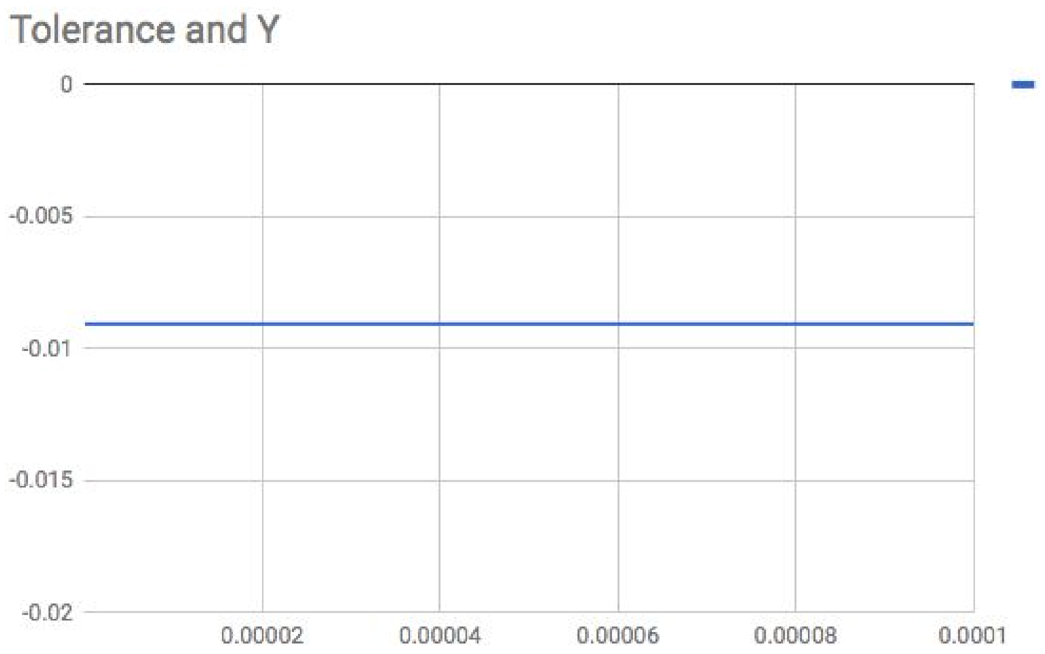


Tolerance and Time



Tolerance and X



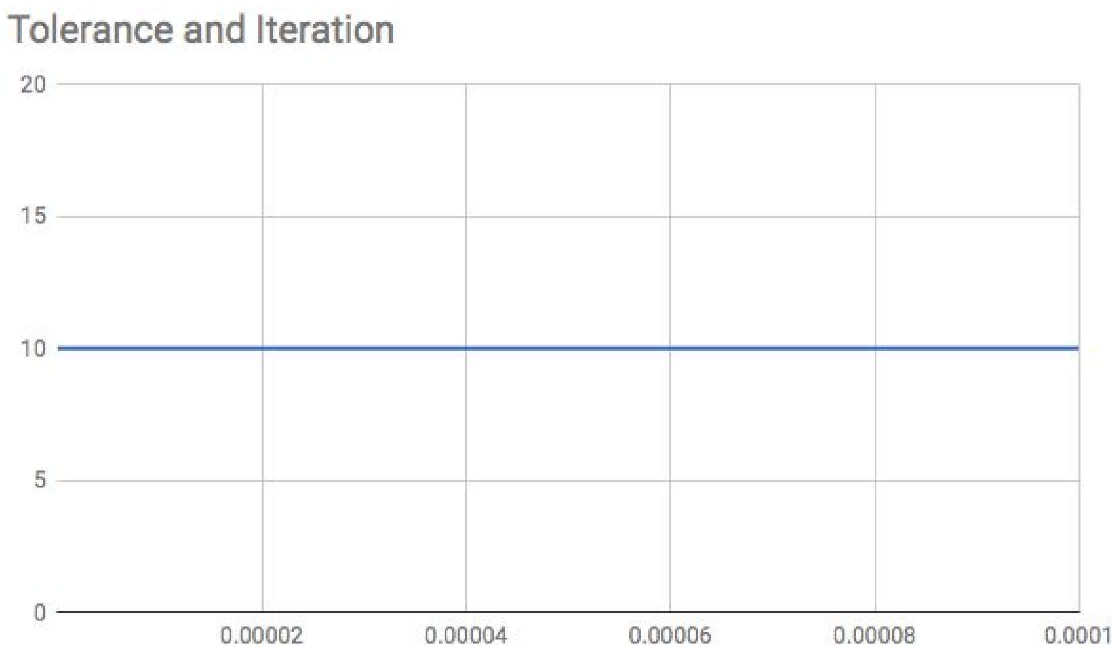


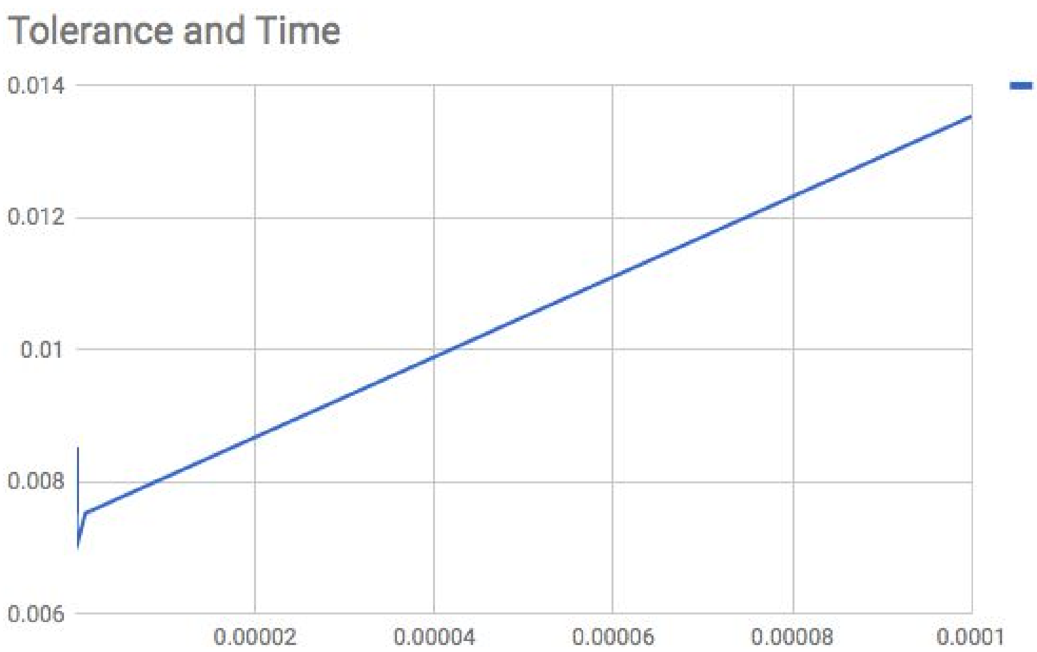
d. Fungsi B

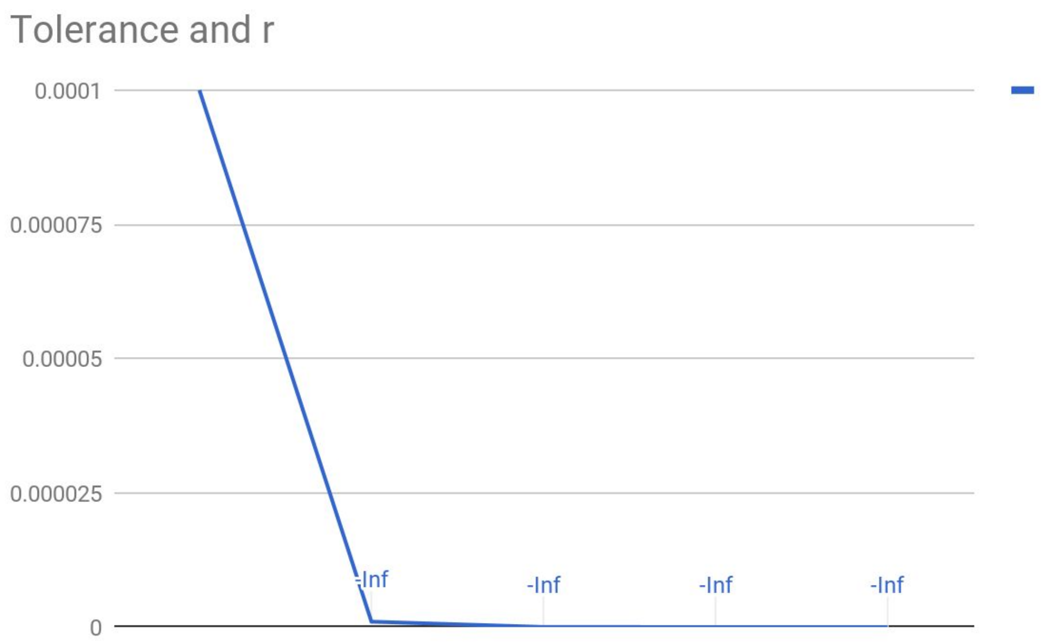
Dari fungsi B, didapatkan hasil eksperimen sebagai berikut.

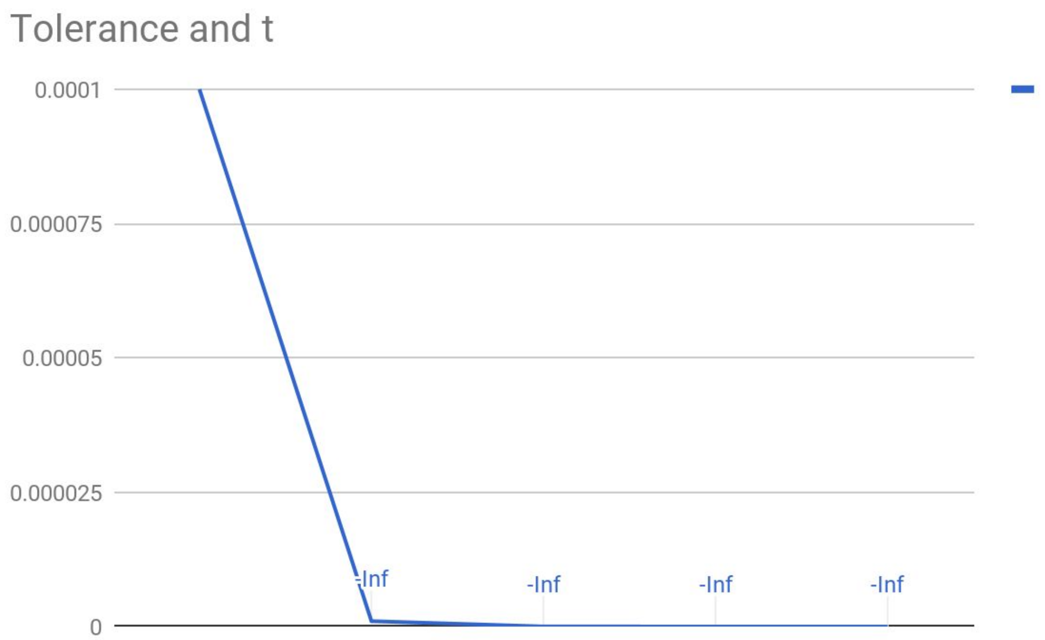
Tol	Iter	time	r	t
0.0001	10	0.013535	-Inf	-Inf
0.000001	10	0.0075200	-Inf	-Inf
0.00000001	10	0.0070190	-Inf	-Inf
1E-10	10	0.0085220	-Inf	-Inf
1E-12	10	0.0075209	-Inf	-Inf

Dari hasil eksperimen tersebut, dapat direpresentasikan kedalam bentuk grafik, yakni









Poblano

1. **Limited-memory BFGS** (**lbfgs**)

e. Fungsi A

Tol	Jumlah Iterasi	FuncEvals	F(x)	||G(x)||/N	Waktu (s)
0.001	2	5	0	0	0.022001
1e-005	2	5	0	0	0.0130009
1e-007	2	5	0	0	0.0140011
1e-009	2	5	0	0	0.015001
1e-011	2	5	0	0	0.0130009

f. Fungsi B

Tol	Jumlah Iterasi	FuncEvals	F(x)	||G(x)||/N	Waktu (s)
0.001	2	21	-1.0217e+50	9.19923396360174e+37	0.0190011
1e-005	2	21	-1.0217e+50	9.19923396360174e+37	0.023002

1e-007	2	21	-1.0217e+50	9.19923396360174e+37	0.0190009
1e-009	2	21	-1.0217e+50	9.19923396360174e+37	0.019001
1e-011	2	21	-1.0217e+50	9.19923396360174e+37	0.0190011

g. Fungsi C (Griewank)

Jumlah Parameter	**Tol**	**Jumlah Iterasi**	**FuncEvals**	**F(x)**	**||G(x)||/N**	**Waktu (s)**
8	0.001	18	70	0.09352512	0.00000928	0.348019
	1e-005	18	70	0.09352512	0.00000928	0.417024
	1e-007	18	70	0.09352512	0.00000928	0.35002
	1e-009	18	70	0.09352512	0.00000928	0.356021
	1e-011	18	70	0.09352512	0.00000928	0.34602
8^2	0.001	17	55	0.00000002	0.00000427	1.11106

	1e-005	17	55	0.00000002	0.00000427	0.862049
	1e-007	17	55	0.00000002	0.00000427	0.851048
	1e-009	17	55	0.00000002	0.00000427	0.850049
	1e-011	17	55	0.00000002	0.00000427	0.848048
8^3	0.001	8	25	0.00000005	0.00000434	0.807046
	1e-005	8	25	0.00000005	0.00000434	0.814046
	1e-007	8	25	0.00000005	0.00000434	0.809046
	1e-009	8	25	0.00000005	0.00000434	0.844049
	1e-011	8	25	0.00000005	0.00000434	0.846048
8^4	0.001	1	8	0	0.00000002	0.443025
	1e-005	1	8	0	0.00000002	0.475027
	1e-007	1	8	0	0.00000002	0.444026
	1e-009	1	8	0	0.00000002	0.446025
	1e-01	1	8	0	0.00000002	0.448025

	1					
8^5	0.001	1	8	0	0	0.685039
	1e-005	1	8	0	0	0.694039
	1e-007	1	8	0	0	0.711041
	1e-009	1	8	0	0	0.677039
	1e-011	1	8	0	0	0.679038

2. **Nonlinear Conjugate Gradient** (**ncg**)

   h. Fungsi A

Tol	**Jumlah Iterasi**	**FuncEvals**	**F(x)**	**||G(x)||/N**	**Waktu (s)**
0.001	3	4	0.00002036	0.00320313	0.015001
1e-005	3	4	0.00002036	0.00320313	0.0140009
1e-007	3	4	0.00002036	0.00320313	0.015001
1e-009	3	4	0.00002036	0.00320313	0.0200019
1e-011	3	4	0.00002036	0.00320313	0.0140009

i. Fungsi B

Tol	Jumlah Iterasi	FuncEvals	F(x)	||G(x)||/N	Waktu (s)
0.001	3	61	-Inf	NaN	0.0350021
1e-005	3	61	-Inf	NaN	0.0350021
1e-007	3	61	-Inf	NaN	0.0390019
1e-009	3	61	-Inf	NaN	0.037002
1e-011	3	61	-Inf	NaN	0.039002

j. Fungsi C (Griewank)

Jumlah Parameter	Tol	Jumlah Iterasi	FuncEvals	F(x)	||G(x)||/N	Waktu (s)
8	0.001	17	80	0.07633810	0.00000854	0.461027
	1e-005	17	80	0.07633810	0.00000854	0.390022
	1e-00	17	80	0.0763381	0.00000854	0.392022

	1e-007			0		
	1e-009	17	80	0.07633810	0.00000854	0.388022
	1e-011	17	80	0.07633810	0.00000854	0.396022
8^2	0.001	18	57	0.00000003	0.00000761	0.999057
	1e-005	18	57	0.00000003	0.00000761	0.915053
	1e-007	18	57	0.00000003	0.00000761	0.884051
	1e-009	18	57	0.00000003	0.00000761	0.87405
	1e-011	18	57	0.00000003	0.00000761	0.893051
8^3	0.001	9	24	0.00000011	0.00000647	0.790046
	1e-005	9	24	0.00000011	0.00000647	0.780045
	1e-007	9	24	0.00000011	0.00000647	0.812046
	1e-009	9	24	0.00000011	0.00000647	0.805046
	1e-011	9	24	0.00000011	0.00000647	0.803046
8^4	0.001	1	8	0	0.00000002	0.448026

	1e-005	1	8	0	0.00000002	0.457026
	1e-007	1	8	0	0.00000002	0.485028
	1e-009	1	8	0	0.00000002	0.450026
	1e-011	1	8	0	0.00000002	0.449026
8^5	0.001	1	8	0	0	0.719042
	1e-005	1	8	0	0	0.686039
	1e-007	1	8	0	0	0.679039
	1e-009	1	8	0	0	0.680039
	1e-011	1	8	0	0	0.793046

2.3 Analisis

2.31 Analisis Teknis

Steepest Descent :

Processor : intel core i7, 3.1 Ghz
Memory : 8Gb
OS : Ubuntu 16.04 , 64 bit

Newton :

Processor : intel core i7, 3.1 Ghz

Memory : 8Gb

OS : Windows 10, 64 bit

Quasi-Newton :

Processor : intel core i7, 3.1 Ghz
Memory : 8Gb
OS : Windows 10, 64 bit

Poblano :

Processor : intel core i7, 3.1 Ghz
Memory : 8Gb
OS : Ubuntu 16.04, 64 bit

2.32 Analisis Hasil

Steepest Descent :

Berdasarkan eksperimen yang kami lakukan steepest descent memiliki grafik yang linear terhadap toleransi. Pada program yang dijalankan tingkat konvergensi sangat bergantung pada nilai toleransi dan initial awal. Apabila memiih nilai initial yang mendekati solusi hasil maka akan mendapatkan konvergensi yang lebih cepat. Nilai toleransi yang besar menyebabkan solusi hasil lebih presisi namun lebih lama.

Pada algoritma Steepest descent pada fungsi berkendala, tidak didapatkan solusi karena kesalahan pada pemilihan nilai awal. Selain itu bisa jadi solusi akhir stuck pada local minima. Pada kode program juga seharusnya di cek turunan kedua dari fungsi tidak hanya turunan pertama, karena turunan kedua menentukan apakah sebuah fungsi memiliki nilai minimum atau maksimum sedangkan turunan pertama hanya mengecek apakah fungsi tersebut merupakan titik kritis atau bukan.

Newton :

Quasi Newton :

Berdasarkan eksperimen yang kami lakukan, Quasi Newton memiliki grafik yang linear terhadap toleransi. Berbeda dengan metode sebelumnya yaitu steepest descent, tingkat konvergensinya tidak bergantung pada nilai toleransi dan initial awal. Karena quasi newton bergantung pada pemilihan B. Oleh karena itulah, kecepatan untuk mencapai konvergensi sangat cepat berdasarkan eksperimen yang telah dilakukan.

Poblano :

Berdasarkan eksperimen tersebut, dapat dilihat bahwa waktu eksekusi yang dihasilkan lebih singkat dibandingkan dengan waktu hasil eksekusi dari soal 1. Untuk soal 1. b. tidak ditemukan hasil dikarenakan nilai ||G(x)||/N adalah NaN. Hasil dari LBFGS dan NCG pun tidak jauh berbeda. Terdapat nilai F yang membuat iterasi yang dilakukan tidak sebanyak iterasi yang dihasilkan oleh soal 1.

BAB 3 PENUTUP

3.1 Kesimpulan

Secara teori, metode quasi newton adalah metode yang paling baik dibandingkan metode newton dan steepest descent. Jika ditinjau dari kecepatan konvergensi, metode quasi newton memiliki kecepatan super linier (1, ...), metode newton memiliki kecepatan kuadratik (2), sementa metode steepest descent memiliki kecepatan linier (n).

Berdasarkan hasil pengamatan, didapatkan hasil bahwa metode quasi newton memang merupakan metode yang paling baik jika ditinjau dari kecepatan konvergensinya jika dibandingkan dengan metode steepest descent dan newton.

3.2 Log Diskusi

Diskusi 1

Hari / Tanggal : Rabu, 25 April 2018

Tempat : Melalui Line app

Waktu :1.00 WIB

Agenda Diskusi : Pembagian Tugas
Capaian Diskusi :Masing-masing anggota mendapatkan tugas yang sudah di-*assign*

Diskusi 2

Hari / Tanggal : Selasa, 8 Mei 2018
Tempat : Melalui *LINE app*
Waktu : 12.30 - 15.00 WIB
Agenda Diskusi : Membahas keunikan masing-masing metode
Capaian Diskusi : Mendapatkan *sharing knowledge* tentang masing -masing metode yang telah diimplementasikan

3.3 Pembagian Kerja

No	Nama	Tugas	Persentase (%)
1.	Fachrur Rozi	Newton Method	20%
2.	Faridah Nur Suci A	BFGS	20%
3.	Jessica Naraiswari A	Quasi Newton Method	20%
4.	Novianti Aliasih	Poblano (No 2)	20%
5.	Nur Intan	Steepest Descent Method	20%

DAFTAR PUSTAKA

Basaruddin, T. (2007), Komputasi Numerik, Fakultas Ilmu Komputer, Jakarta.

Optimisasi non linier. (2018, 8 Mei). Diambil dari https://scele.cs.ui.ac.id/pluginfile.php/34962/mod_resource/content/3/Lecture-5_anum_NonLinear_Opt_18_upd240418.pptx

Poblano Toolbox V1.1 Documentation.

LAMPIRAN