

Solution to Assignment 5

I will appreciate it if you could give me some advice on my assignment!

December 26, 2018

phd exercise

- Given the standard form of LP problem

$$\begin{array}{ll} \min & \mathbf{c}^T \mathbf{x} \\ \text{such that} & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array}$$

with $\mathbf{A} \in \mathbb{R}^{m \times n}$, $m < n$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^n$, we apply the pdipm to solve this problem, i.e., solve the linear system (1):

$$F(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \begin{pmatrix} \mathbf{A}^T \mathbf{y} + \mathbf{z} - \mathbf{c} \\ \mathbf{A}\mathbf{x} - \mathbf{b} \\ \mathbf{x} \circ \mathbf{z} \end{pmatrix} = \mathbf{0}, \quad (\mathbf{x}, \mathbf{z}) \geq 0, \quad (1)$$

which is also the *optimality conditions* of the standard form LP. Prove that the *Jacobian matrix* $F'(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is non-singular if $(\mathbf{x}, \mathbf{z}) > 0$ and \mathbf{A} has the *full row rank*.

Proof. As have done in last assignment, we obtain:

$$F'(\mathbf{x}, \mathbf{y}, \mathbf{z}) := \mathbf{Y} = \begin{pmatrix} \mathbf{0} & \mathbf{A}^T & \mathbf{I} \\ \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{Z} & \mathbf{0} & \mathbf{X} \end{pmatrix}_{(n+m+n) \times (n+m+n)}$$

with $\mathbf{Z} = \text{diag}(z_1, \dots, z_n)$ and $\mathbf{X} = \text{diag}(x_1, \dots, x_n)$. Given the condition that $\text{rank}(\mathbf{A}) = m < n$ and $(\mathbf{x}, \mathbf{z}) > 0$, it suffices to show that the null space matrix \mathbf{Y} has only trivial solution, i.e., $\mathcal{N}(\mathbf{Y}) = \{\mathbf{0}\}$. Given any $(\mathbf{u}, \mathbf{v}, \mathbf{w}) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n$ in the null space $\mathcal{N}(\mathbf{Y})$, it suffices to show $(\mathbf{u}, \mathbf{v}, \mathbf{w}) = (\mathbf{0}, \mathbf{0}, \mathbf{0})$. From $\mathbf{Y}(\mathbf{u}, \mathbf{v}, \mathbf{w}) = (\mathbf{0}, \mathbf{0}, \mathbf{0})$ we derive:

$$\mathbf{A}^T \mathbf{v} + \mathbf{w} = \mathbf{0} \quad (2a)$$

$$\mathbf{A}\mathbf{u} = \mathbf{0} \quad (2b)$$

$$\mathbf{Z}\mathbf{u} + \mathbf{X}\mathbf{w} = \mathbf{0} \quad (2c)$$

- Left-multiplying (2a) with \mathbf{u}^T and noting (2b), we derive:

$$\mathbf{u}^T \mathbf{w} = 0 \quad (3)$$

We re-write (2c) as:

$$\mathbf{u} + \mathbf{Z}^{-1} \mathbf{X} \mathbf{w} = 0 \quad (4a)$$

$$\mathbf{w} + \mathbf{X}^{-1} \mathbf{Z} \mathbf{u} = 0 \quad (4b)$$

Combining with (3), equivalently we have

$$\mathbf{w}^T \mathbf{Z}^{-1} \mathbf{X} \mathbf{w} = 0 \quad (5a)$$

$$\mathbf{u}^T \mathbf{X}^{-1} \mathbf{Z} \mathbf{u} = 0 \quad (5b)$$

Note that $\mathbf{Z}, \mathbf{X} \succ 0$, i.e., $\mathbf{Z}^{-1} \mathbf{X}, \mathbf{X}^{-1} \mathbf{Z} \succ 0$, which implies $\mathbf{w} = \mathbf{u} = \mathbf{0}$.

- Finally, consider that $\mathbf{w} = \mathbf{0}$ and (2a), we derive $\mathbf{A}^T \mathbf{v} = \mathbf{0}$. Since $\text{rank}(\mathbf{A}^T) = m$, i.e., columns of \mathbf{A}^T are linearly independent, we derive $\mathbf{v} = \mathbf{0}$.

The proof is complete. \square

2. Let x^* be a feasible point that is *regular* and together with some λ^* satisfies the first and second order necessary conditions:

$$\nabla f(x^*) + \sum_{i=1}^m \lambda_i^* \nabla h_i(x) = 0 \iff \nabla_x L(x^*, \lambda^*) = 0 \quad (6a)$$

$$y^T \left[\nabla^2 f(x^*) + \sum_{i=1}^m \lambda_i^* \nabla^2 h_i(x) \right] y \geq 0 \iff y^T \nabla_{xx}^2 L(x^*, \lambda^*) y \geq 0, \quad \text{for all } y \neq 0 \text{ with } \langle \nabla h(x^*), y \rangle = 0 \quad (6b)$$

Show that x^* and λ^* satisfy the sufficient conditions

$$\nabla_x L(x^*, \lambda^*) = 0, \quad \nabla_\lambda L(x^*, \lambda^*) = 0, \quad (7a)$$

$$y^T \nabla_{xx}^2 L(x^*, \lambda^*) y > 0, \quad \text{for all } y \neq 0 \text{ with } \langle \nabla h(x^*), y \rangle = 0 \quad (7b)$$

if and only if the matrix

$$\mathbf{R} = \begin{pmatrix} \nabla_{xx}^2 L(x^*, \lambda^*) & \nabla h(x^*) \\ \nabla^T h(x^*) & 0 \end{pmatrix}$$

is non-singular.

Proof. Necessity. To show the forward assertion, assume \mathbf{R} is singular, i.e., there exists nonzero vector (w, v) that is in the null space of \mathbf{R} , i.e.,

$$\nabla_{xx}^2 L(x^*, \lambda^*) w + \nabla h(x^*) v = 0 \quad (8a)$$

$$\langle \nabla h(x^*), w \rangle = 0 \quad (8b)$$

- Left-multiplying (8a) with w^T , and consider (11b), we derive:

$$w^T \nabla_{xx}^2 L(x^*, \lambda^*) w = 0 \quad (9)$$

As $\langle \nabla h(x^*), w \rangle = 0$, consider (7b), we derive $w = 0$. Substituting w into (8a), we derive $v = 0$, since $\nabla h_1(x^*), \dots, \nabla h_m(x^*)$ is linearly independent. Thus we derive a contradiction. Hence, \mathbf{R} must be non-singular. which follows that $v = 0$.

Sufficiency. Assume that there exists $\bar{y} \neq 0$ such that $\langle \nabla h(x^*), \bar{y} \rangle = 0$ but $\bar{y}^T \nabla_{xx}^2 L(x^*, \lambda^*) \bar{y} = 0$ (From (6b), note that $\bar{y}^T \nabla_{xx}^2 L(x^*, \lambda^*) \bar{y} \geq 0$ for any $\bar{y} \neq 0$ such that $\langle \nabla h(x^*), \bar{y} \rangle = 0$). Thus \bar{y} is essentially the local minimum for the optimization problem

$$\min_{\text{such that } \langle \nabla h(x^*), y \rangle = 0} y^T \nabla_{xx}^2 L(x^*, \lambda^*) y \quad (10)$$

Consider the Lagrange function $G(y, \mu) = y^T \nabla_{xx}^2 L(x^*, \lambda^*) y + \mu \langle \nabla h(x^*), y \rangle$, applying the first order necessary condition for constraint problem, we have

$$\nabla_y G(\bar{y}, \bar{\mu}) = 2 \nabla_{xx}^2 L(x^*, \lambda^*) \bar{y} + \bar{\mu}^* \nabla h(x^*) = 0.$$

Combining with the condition $\langle \nabla h(x^*), \bar{y} \rangle = 0$, we obtain a matrix-form system:

$$\begin{pmatrix} \nabla_{xx}^2 L(x^*, \lambda^*) & \nabla h(x^*) \\ \nabla^T h(x^*) & 0 \end{pmatrix} \begin{pmatrix} 2\bar{y} \\ \bar{\mu} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Therefore, $(2\bar{y}, \bar{\mu}) \in \mathcal{N}(\mathbf{R})$ but $\bar{y} \neq 0$, i.e., the null space of \mathbf{R} contains a non-trivial solution, which is a contradiction.

The proof is complete. □

3. Let x^* be a feasible point that is *regular* and together with some λ^* satisfies the first and second order necessary conditions:

$$\nabla_x L(x^*, \lambda^*) = 0, \quad \nabla_\lambda L(x^*, \lambda^*) = 0, \quad (11a)$$

$$y^T \nabla_{xx}^2 L(x^*, \lambda^*) y > 0, \quad \text{for all } y \neq 0 \text{ with } \langle \nabla h(x^*), y \rangle = 0 \quad (11b)$$

then for some u and corresponding $\lambda(u)$ in the open sphere S centered at $u = 0$, for every scalar c for which the matrix

$$A_c(u) = \nabla_{xx}^2 L(x(u), \lambda(u)) + c \nabla h(x(u)) \nabla^T h(x(u)) \quad (12)$$

is invertible, we have

$$\nabla^2 p(u) = (\nabla^T h(x(u)) A_c(u)^{-1} \nabla h(x(u)))^{-1} - cI \quad (13)$$

Proof. From the sensitivity theorem, we have

$$\nabla f(x(u)) + \nabla h(x(u)) \lambda(u) = 0 \quad (14a)$$

$$h(x(u)) = u \quad (14b)$$

Differentiating (14a) and (14b) both sides w.r.t. u , we have:

$$\nabla x(u) \underbrace{[\nabla^2 f(x(u)) + \lambda^T \nabla^2 h(x(u))]}_{\nabla_{xx}^2 L(x(u), \lambda(u))} + \nabla \lambda(u) \nabla^T h(x(u)) = 0 \quad (15a)$$

$$\nabla x(u) \nabla h(x(u)) = I \quad (15b)$$

Right-multiplying $\nabla^T h(x(u))$ both sides on (15b), we obtain:

$$\nabla x(u) \nabla h(x(u)) \nabla^T h(x(u)) = \nabla^T h(x(u)) \quad (15c)$$

Adding (15a) with $c \times (15c)$, we see that

$$\nabla x(u) [\nabla_{xx}^2 L(x(u), \lambda(u)) + c \nabla h(x(u)) \nabla^T h(x(u))] + (\nabla \lambda(u) - cI) \nabla^T h(x(u)) = 0 \quad (16)$$

For every c s.t. the inverse of $A_c(u)$ exists, we rightmultiply (16) with $A_c(u)$:

$$\nabla x(u) + (\nabla \lambda(u) - cI) \nabla^T h(x(u)) [\nabla_{xx}^2 L(x(u), \lambda(u)) + c \nabla h(x(u)) \nabla^T h(x(u))]^{-1} = 0 \quad (17)$$

Left-multiplying (17) with $\nabla h(x(u))$ and using equation (15b), we derive:

$$I + \nabla h(x(u)) (\nabla \lambda(u) - cI) \nabla^T h(x(u)) [\nabla_{xx}^2 L(x(u), \lambda(u)) + c \nabla h(x(u)) \nabla^T h(x(u))]^{-1} = 0$$

Or equivalently,

$$-\nabla \lambda(u) = (\nabla^T h(x(u)) A_c(u)^{-1} \nabla h(x(u)))^{-1} - cI$$

Differentiating $\nabla p(u) = -\lambda(u)$ both sides, we derive that

$$\nabla^2 p(u) = -\nabla \lambda(u) = (\nabla^T h(x(u)) A_c(u)^{-1} \nabla h(x(u)))^{-1} - cI$$

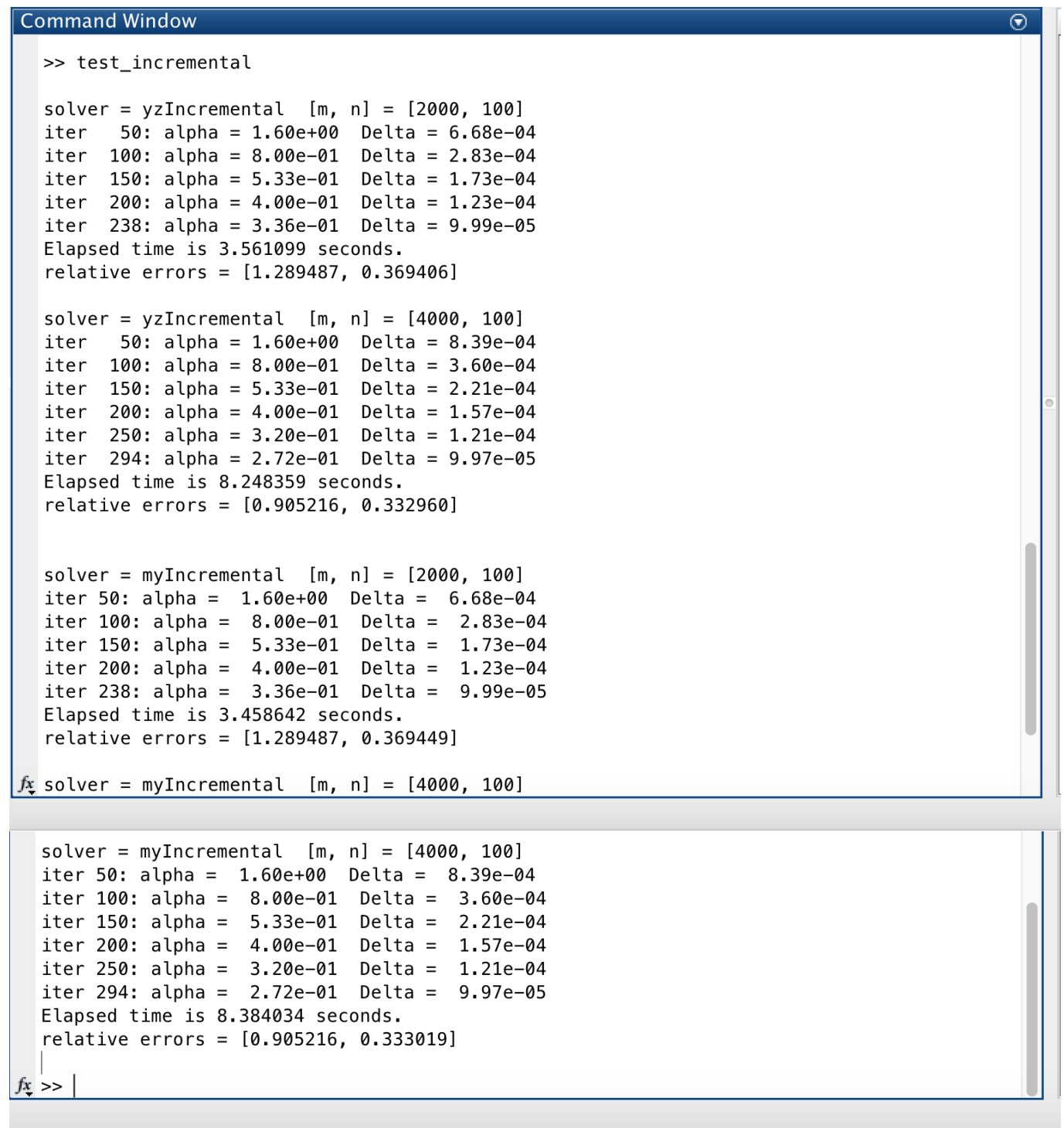
□

MATLAB Project: Incremental Gradient Method

A copy of my code myIncremental.m

```
function x = myIncremental(A,b,x0,tol,maxit)
% Input:
%   A: given coefficient matrix
%   b: given cost vector
%   x0: initial guess
%   tol: tolerance
%   maxit: maximum iteration
% Output:
%   x: least squares solution
[m,~] = size(A);
x = x0;
theta = 80;
g = @(ai,x,bi)(ai*(ai'*x-bi));
A = A';
for iter = 1:maxit
    alpha = theta/iter;
    psi = x;
    for j = 1:m
        ai = A(:, j);
        bi = b(j);
        psi = psi - alpha * g(ai,psi,bi);
    end
    %psi = psi - alpha * A' * (A*psi - b);
    Delta = norm(x-psi)/ norm(x);
    if Delta <= tol
        fprintf('iter %2i: alpha = %9.2e Delta = %9.2e\n',...
            iter,alpha, Delta)
        break;
    end
    if mod(iter,50) == 0
        fprintf('iter %2i: alpha = %9.2e Delta = %9.2e\n',...
            iter,alpha, Delta);
    end
    x = psi;
end
end
```

MATLAB Screen Printout for test incremental



The image shows a MATLAB Command Window with a blue title bar and a white background. It displays the output of a script named 'test_incremental'. The output is organized into three sections, each starting with a solver assignment and matrix dimensions. The first section uses 'yzIncremental' for a 2000x100 matrix, showing iterations 50 to 238, an elapsed time of 3.561099 seconds, and relative errors. The second section uses 'yzIncremental' for a 4000x100 matrix, showing iterations 50 to 294, an elapsed time of 8.248359 seconds, and relative errors. The third section uses 'myIncremental' for a 2000x100 matrix, showing iterations 50 to 238, an elapsed time of 3.458642 seconds, and relative errors. A fourth line of code for 'myIncremental' on a 4000x100 matrix is shown but not executed. The Command Window has a scrollbar on the right and a status bar at the bottom.

```
Command Window

>> test_incremental

solver = yzIncremental [m, n] = [2000, 100]
iter 50: alpha = 1.60e+00 Delta = 6.68e-04
iter 100: alpha = 8.00e-01 Delta = 2.83e-04
iter 150: alpha = 5.33e-01 Delta = 1.73e-04
iter 200: alpha = 4.00e-01 Delta = 1.23e-04
iter 238: alpha = 3.36e-01 Delta = 9.99e-05
Elapsed time is 3.561099 seconds.
relative errors = [1.289487, 0.369406]

solver = yzIncremental [m, n] = [4000, 100]
iter 50: alpha = 1.60e+00 Delta = 8.39e-04
iter 100: alpha = 8.00e-01 Delta = 3.60e-04
iter 150: alpha = 5.33e-01 Delta = 2.21e-04
iter 200: alpha = 4.00e-01 Delta = 1.57e-04
iter 250: alpha = 3.20e-01 Delta = 1.21e-04
iter 294: alpha = 2.72e-01 Delta = 9.97e-05
Elapsed time is 8.248359 seconds.
relative errors = [0.905216, 0.332960]

solver = myIncremental [m, n] = [2000, 100]
iter 50: alpha = 1.60e+00 Delta = 6.68e-04
iter 100: alpha = 8.00e-01 Delta = 2.83e-04
iter 150: alpha = 5.33e-01 Delta = 1.73e-04
iter 200: alpha = 4.00e-01 Delta = 1.23e-04
iter 238: alpha = 3.36e-01 Delta = 9.99e-05
Elapsed time is 3.458642 seconds.
relative errors = [1.289487, 0.369449]

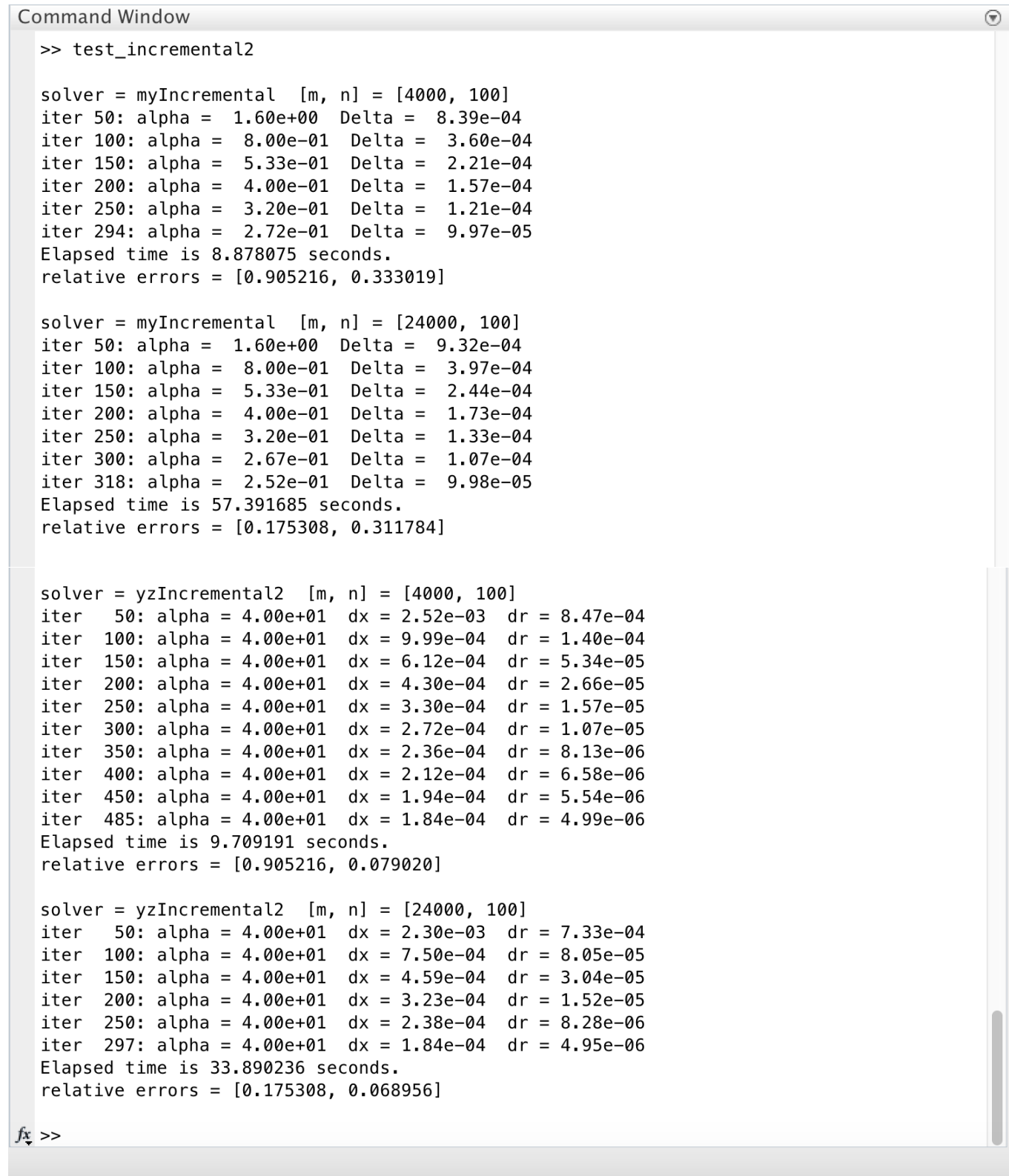
fx solver = myIncremental [m, n] = [4000, 100]

solver = myIncremental [m, n] = [4000, 100]
iter 50: alpha = 1.60e+00 Delta = 8.39e-04
iter 100: alpha = 8.00e-01 Delta = 3.60e-04
iter 150: alpha = 5.33e-01 Delta = 2.21e-04
iter 200: alpha = 4.00e-01 Delta = 1.57e-04
iter 250: alpha = 3.20e-01 Delta = 1.21e-04
iter 294: alpha = 2.72e-01 Delta = 9.97e-05
Elapsed time is 8.384034 seconds.
relative errors = [0.905216, 0.333019]

fx >> |
```

Figure 1: Matlab Screen Printout from file test_incremental

MATLAB Screen Printout for test_incremental2



```
Command Window

>> test_incremental2

solver = myIncremental [m, n] = [4000, 100]
iter 50: alpha = 1.60e+00 Delta = 8.39e-04
iter 100: alpha = 8.00e-01 Delta = 3.60e-04
iter 150: alpha = 5.33e-01 Delta = 2.21e-04
iter 200: alpha = 4.00e-01 Delta = 1.57e-04
iter 250: alpha = 3.20e-01 Delta = 1.21e-04
iter 294: alpha = 2.72e-01 Delta = 9.97e-05
Elapsed time is 8.878075 seconds.
relative errors = [0.905216, 0.333019]

solver = myIncremental [m, n] = [24000, 100]
iter 50: alpha = 1.60e+00 Delta = 9.32e-04
iter 100: alpha = 8.00e-01 Delta = 3.97e-04
iter 150: alpha = 5.33e-01 Delta = 2.44e-04
iter 200: alpha = 4.00e-01 Delta = 1.73e-04
iter 250: alpha = 3.20e-01 Delta = 1.33e-04
iter 300: alpha = 2.67e-01 Delta = 1.07e-04
iter 318: alpha = 2.52e-01 Delta = 9.98e-05
Elapsed time is 57.391685 seconds.
relative errors = [0.175308, 0.311784]

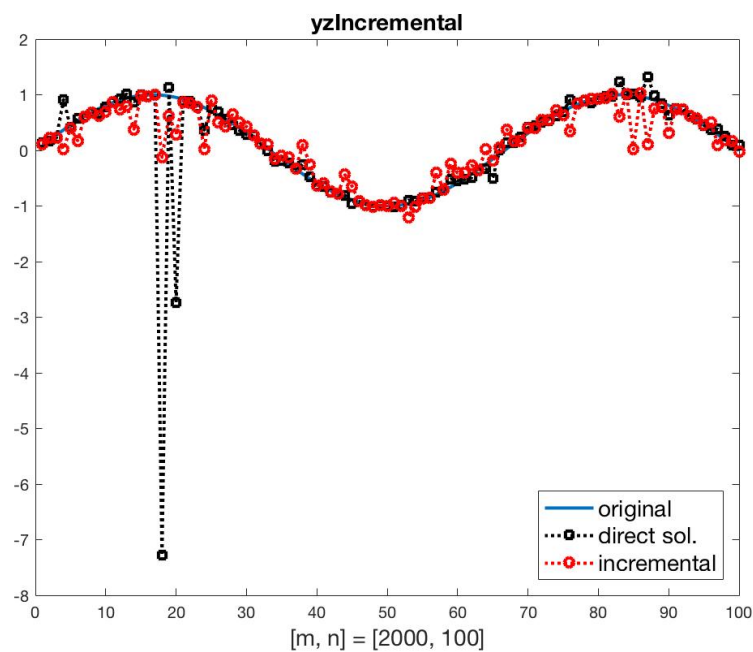
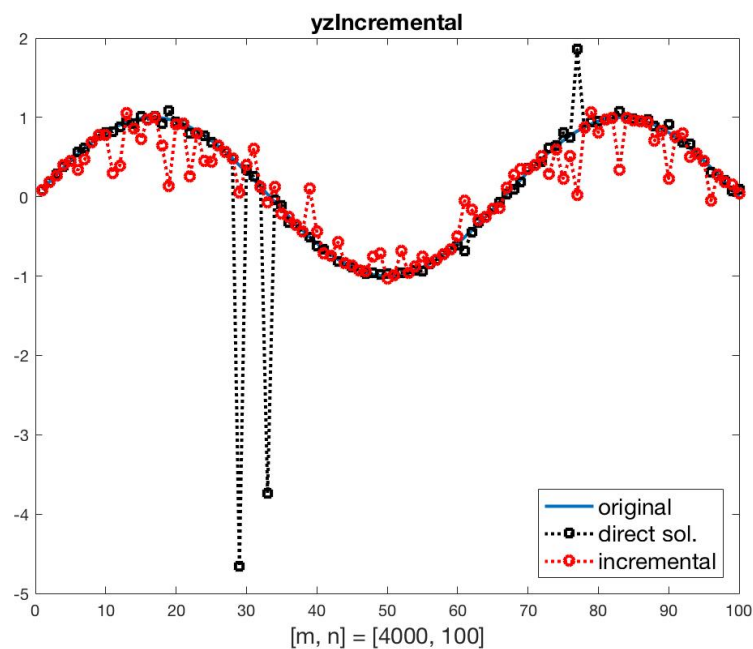
solver = yzIncremental2 [m, n] = [4000, 100]
iter 50: alpha = 4.00e+01 dx = 2.52e-03 dr = 8.47e-04
iter 100: alpha = 4.00e+01 dx = 9.99e-04 dr = 1.40e-04
iter 150: alpha = 4.00e+01 dx = 6.12e-04 dr = 5.34e-05
iter 200: alpha = 4.00e+01 dx = 4.30e-04 dr = 2.66e-05
iter 250: alpha = 4.00e+01 dx = 3.30e-04 dr = 1.57e-05
iter 300: alpha = 4.00e+01 dx = 2.72e-04 dr = 1.07e-05
iter 350: alpha = 4.00e+01 dx = 2.36e-04 dr = 8.13e-06
iter 400: alpha = 4.00e+01 dx = 2.12e-04 dr = 6.58e-06
iter 450: alpha = 4.00e+01 dx = 1.94e-04 dr = 5.54e-06
iter 485: alpha = 4.00e+01 dx = 1.84e-04 dr = 4.99e-06
Elapsed time is 9.709191 seconds.
relative errors = [0.905216, 0.079020]

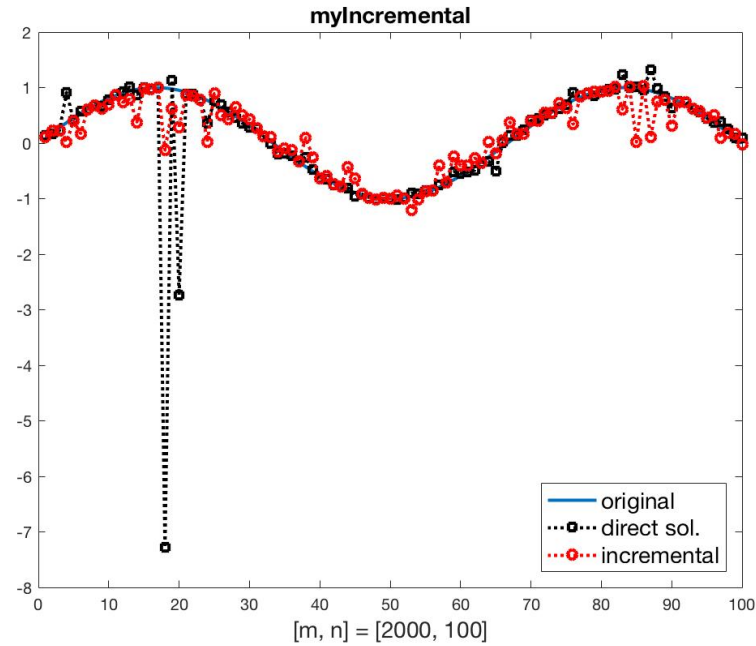
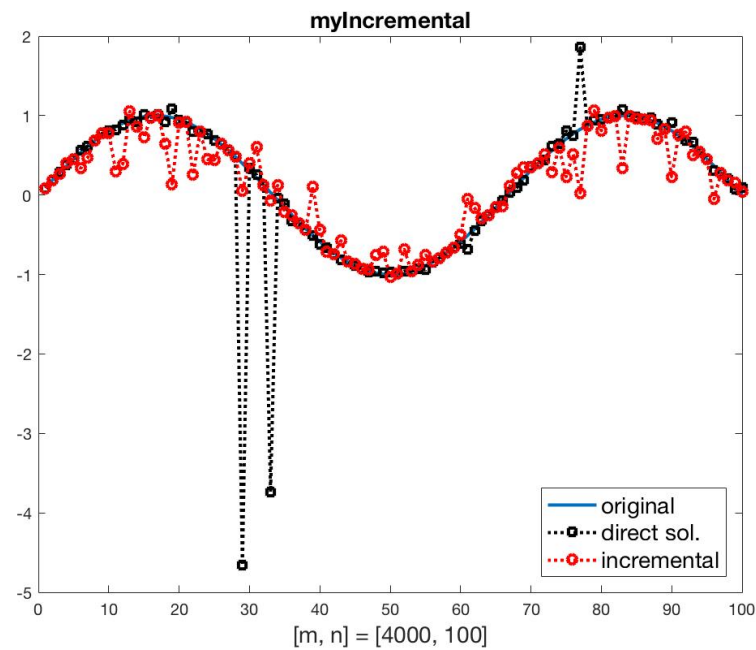
solver = yzIncremental2 [m, n] = [24000, 100]
iter 50: alpha = 4.00e+01 dx = 2.30e-03 dr = 7.33e-04
iter 100: alpha = 4.00e+01 dx = 7.50e-04 dr = 8.05e-05
iter 150: alpha = 4.00e+01 dx = 4.59e-04 dr = 3.04e-05
iter 200: alpha = 4.00e+01 dx = 3.23e-04 dr = 1.52e-05
iter 250: alpha = 4.00e+01 dx = 2.38e-04 dr = 8.28e-06
iter 297: alpha = 4.00e+01 dx = 1.84e-04 dr = 4.95e-06
Elapsed time is 33.890236 seconds.
relative errors = [0.175308, 0.068956]

fx >>
```

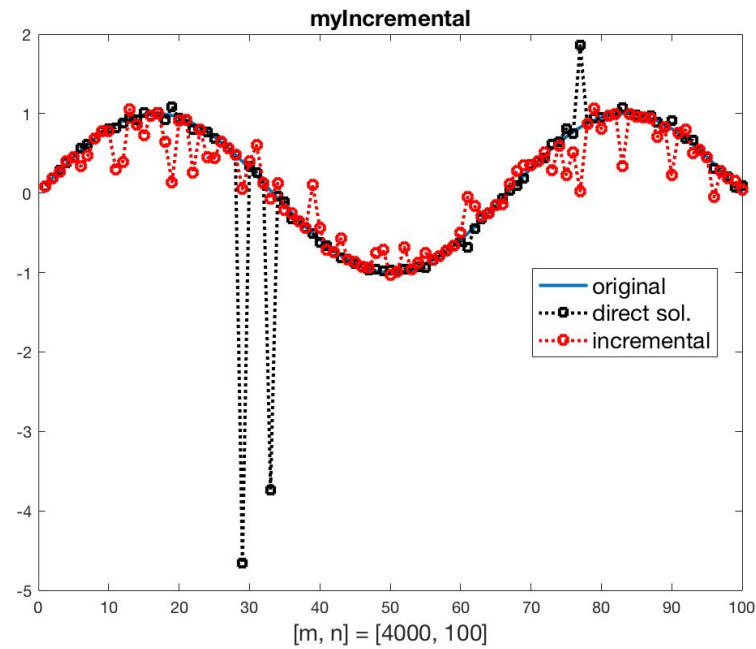
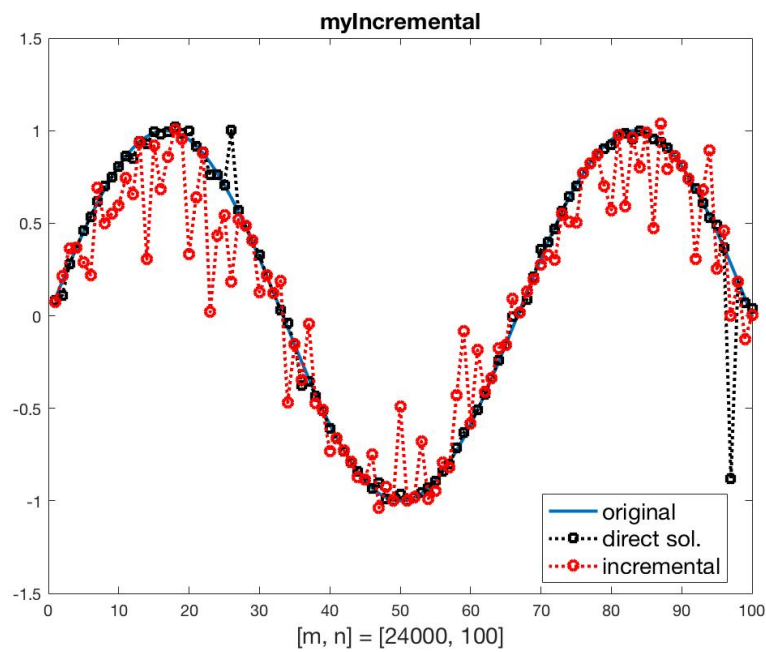
Figure 2: Matlab Screen Printout from file test_incremental2

Matlab Output figures for test incremental

Figure 3: Matlab Output figures from yzIncremental with $(m, n) = (2000, 100)$ Figure 4: Matlab Output figures from yzIncremental with $(m, n) = (4000, 100)$

Figure 5: Matlab Output figures from myIncremental with $(m, n) = (2000, 100)$ Figure 6: Matlab Output figures from myIncremental with $(m, n) = (4000, 100)$

Matlab Output figures for test incremental2

Figure 7: Matlab Output figures from myIncremental with $(m, n) = (4000, 100)$ Figure 8: Matlab Output figures from myIncremental with $(m, n) = (24000, 100)$

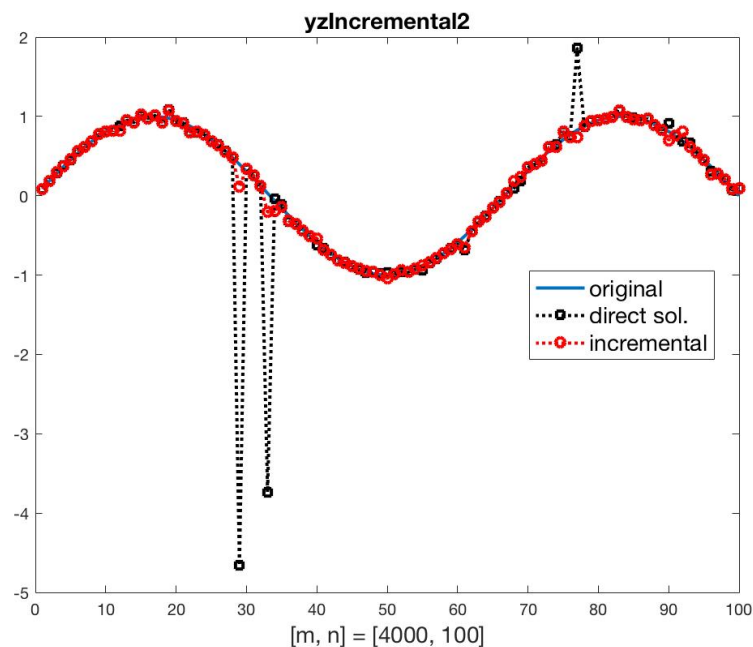


Figure 9: Matlab Output figures from yzIncremental2 with $(m, n) = (4000, 100)$

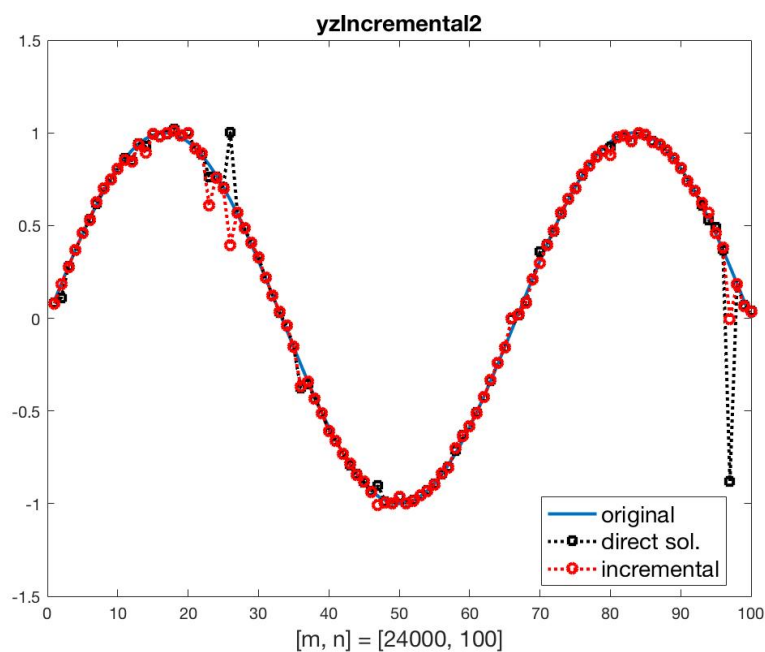


Figure 10: Matlab Output figures from yzIncremental2 with $(m, n) = (24000, 100)$

Short Summary

Introduction This project aims to solve the least squares problem by taking descent directions on each sub-functions iteratively in each iteration.

Save necessity before computation In each iteration we are required to update our \mathbf{x} with

$$\mathbf{x} \leftarrow \mathbf{x} - \alpha * \nabla f_i(x) = \mathbf{x} - \alpha * (a_i * (a_i^T \mathbf{x} - b_i))$$

for many times, which is expansive since MATLAB spends lots of time calling rows of large matrix \mathbf{A} . However, if we pre-save the rows of \mathbf{A} and write a function handle to replace the update command, the code will run much faster.

Choose Reasonable parameters For example, the diminishing step-size should be relatively large when starting the update, so if choosing tiny step size in the beginning, it is difficult for the code to converge.

Comments about yzincremental2 In the new file yzincremental2, Prof. YZ apply the constant step size, but the results are better compared with that by applying diminishing step size. As he said in the lecture, the insights and reasons behind this is still remained to be found.

MATLAB Project: Primal-dual interior-point method for LP

A copy of my code: my pdipm.m(predictor-corrector algorithm)

```
function [x,y,z,iter] = my_pdipm(A,b,c,tol,maxit,prt)
% Input:
%     A = constraint coefficient matrix
%     b = constraint right-hand side vector
%     c = objective coefficient vector
%     tol = tolerance
%     maxit = maximum number of iterations allowed
%     prt = switch for screen printout (1 on, 0 off)
% Output:
%     x = computed primal solution
%     y = computed dual solution
%     z = computed dual slacks
%     iter = iteration counter
%% parameters setting
%p = symamd(abs(A)*abs(A)');
[m,n]=size(A);
p = symamd(abs(A)*abs(A)');%colamd(A');%symamd(A*A');%colamd(A');
%% Initialize
bigMfac = 100;
bigM = max(max(abs(A)));
bigM = max([norm(b,Inf), norm(c,Inf), bigM]);
x = bigMfac*bigM * ones(n,1); %bigMfac*bigM * ones(n,1);
y = zeros(m,1);
z = x;
%bc = 1 + max(norm(b),norm(c));
bc = 1+[norm(b);norm(c);0];
for iter = 1:maxit
    %% initial setting
    rd = -c + A'*y + z;
    rp = -b + A*x;
    rc = x.*z;
    gap = mean(rc);
    %% Stopping criteria
    bc(3) = abs(b'*y)+1;
    residual = sum([norm(rp);norm(rd);norm(rc)]./bc);
    if residual <= tol, break;end;
    switch prt
        case 1
            fprintf('iter %2i: [primal dual gap] = [%9.2e %9.2e %9.2e]\n',iter,[norm(rp);norm(rd);norm(rc)]./bc);
    end
    %% formulate the linear systems
    d = min(5.e+15,x./z);
    M = A* sparse(1:n,1:n,d) *A'; R = chol(M(p,p));%R = chol(M(p,p),'lower','vector');
    %% Predictor step: Solve for dy
    t1 = x.*rd - rc;
    t2 = -(rp+A*(t1./z));
    dy1 = zeros(m,1);
    dy1(p) = R\ (R'\t2(p));
```

```

%% Predictor step: Solve for dz, dx by back substitutions
dx1 = (t1 + x.*(A'*dy1))./z;
dz1 = (-rc - z.* dx1)./x;
%% Corrector Step
ap = -1/min(min(dx1./x),-1);
ad = -1/min(min(dz1./z),-1);
% centering parameter step
mun = ((x + ap * dx1)' * (z + ap * dz1))/n;
sigma = (mun/gap)^3;
mu = gap * min(.2,sigma);
rd = 0; rp = 0; rc = -mu + dx1.*dz1;
t1 = x.*rd - rc;
t2 = -(rp+A*(t1./z));
dy2 = zeros(m,1);
dy2(p) = R\'(t2(p));
dx2 = (t1 + x.*(A'*dy2))./z;
dz2 = (-rc - z.* dx2)./x;
%% Combined Step
dx = dx1+dx2; dy = dy1 + dy2; dz = dz1+dz2;
tau = max(.9995,1-10*gap);
ap = -1/min(min(dx./x),-1);
ad = -1/min(min(dz./z),-1);
ap = min(tau * ap,1);
ad = min(tau * ad,1);
x = x + ap * dx;
z = z + ad * dz;
y = y + ad * dy;
end
x=full(x); z=full(z); y=full(y);
end

```

A copy of my code: my pdip.m (basic algorithm)

```

function [x,y,z,iter] = my_pdipm(A,b,c,tol,maxit,prt)
% Input:
%       A = constraint coefficient matrix
%       b = constraint right-hand side vector
%       c = objective coefficient vector
%       tol = tolerance
%       maxit = maximum number of iterations allowed
%       prt = switch for screen printout (1 on, 0 off)
% Output:
%       x = computed primal solution
%       y = computed dual solution
%       z = computed dual slacks
%       iter = iteration counter

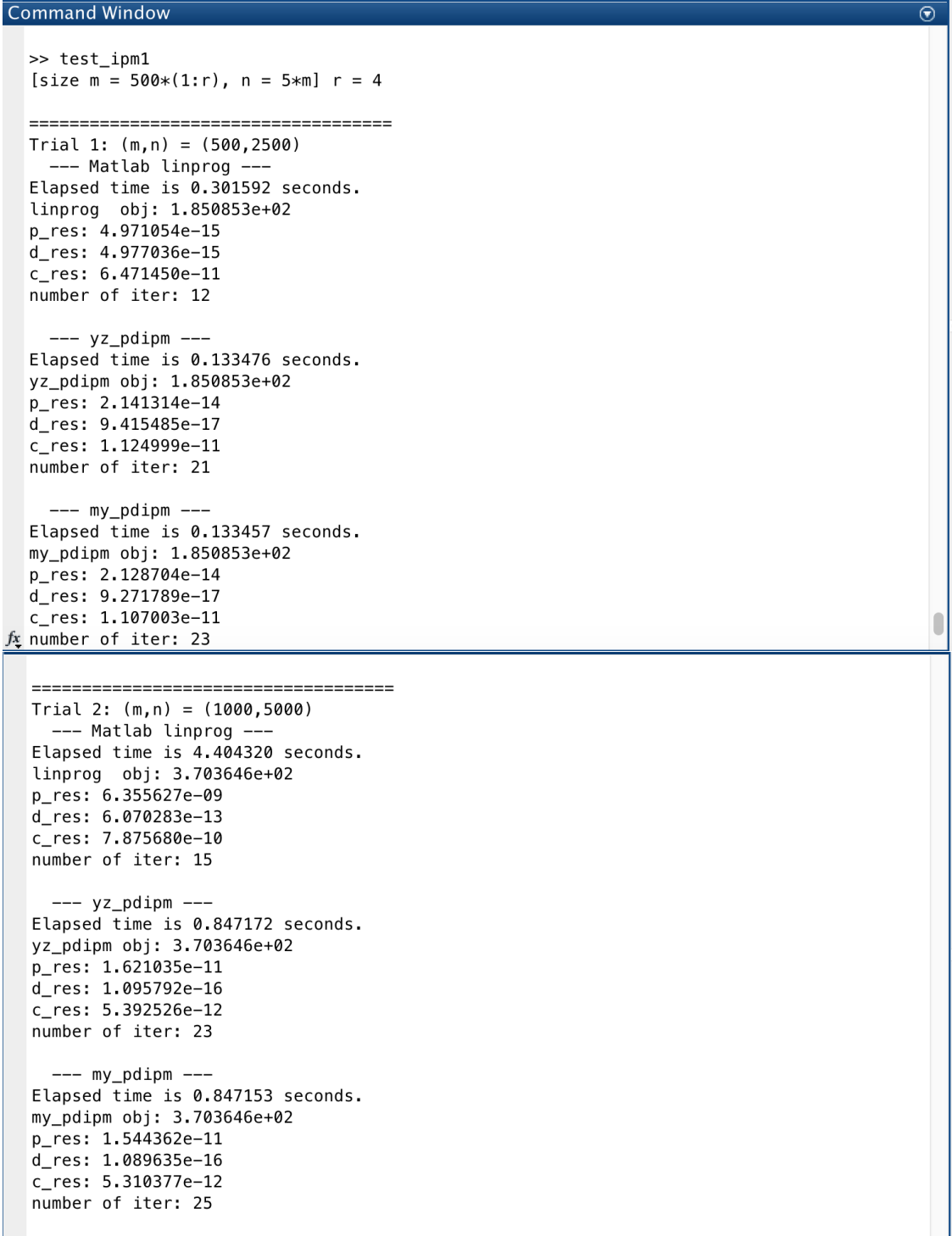
%% parameters setting
%p = symamd(abs(A)*abs(A'))';
[m,n]=size(A);
p = colamd(A');%symamd(A*A');%colamd(A');

```

```

%% Initialize
bigMfac = 100;
bigM = max(max(abs(A)));
bigM = max([norm(b,Inf), norm(c,Inf), bigM]);
x = bigMfac*bigM * ones(n,1);
y = zeros(m,1);
z = x;
%bc = 1 + max(norm(b),norm(c));
bc = 1+[norm(b);norm(c);0];
for iter = 1:maxit
    %% initial setting
    rd = -c + A'*y + z;
    rp = -b + A*x;
    rc = x.*z;
    %% Stopping criteria
    bc(3) = abs(b'*y)+1;
    residual = sum([norm(rp);norm(rd);norm(rc)]./bc);
    if residual <= tol, break;end;
    switch prt
        case 1
            fprintf('iter %2i: [primal dual gap] = [%9.2e %9.2e\n',iter,norm([rd;rp;rc])./bc);
        end
    %% Update rc
    gap = mean(rc);
    rc = rc - min(.1,100*gap)*gap;
    %% Solving for dy
    d = min(5.e+15,x./z);
    M = A* sparse(1:n,1:n,d) *A'; R = chol(M(p,p));%R = chol(M(p,p),'lower','vector');
    t1 = x.*rd - rc;
    t2 = -(rp+A*(t1./z));
    dy = zeros(m,1);
    dy(p) = R\'(R\'t2(p));
    %% Solving for dz, dx by back substitutions
    dx = (t1 + x.*(A'*dy))./z;
    dz = (-rc - z.* dx)./x;
    %% Determine the primal and dual step sizes
    tau = max(.9995,1-gap);
    ap = -1/min(min(dx./x),-1);
    ad = -1/min(min(dz./z),-1);
    ap = min(tau * ap,1);
    ad = min(tau * ad,1);
    x = x + ap * dx;
    z = z + ad * dz;
    y = y + ad * dy;
end
x=full(x); z=full(z); y=full(y);
end

```

MATLAB Screen printout for the run results of test_ipm1.m with $r = 4$ 

```
Command Window

>> test_ipm1
[size m = 500*(1:r), n = 5*m] r = 4

=====
Trial 1: (m,n) = (500,2500)
--- Matlab linprog ---
Elapsed time is 0.301592 seconds.
linprog obj: 1.850853e+02
p_res: 4.971054e-15
d_res: 4.977036e-15
c_res: 6.471450e-11
number of iter: 12

--- yz_pdipm ---
Elapsed time is 0.133476 seconds.
yz_pdipm obj: 1.850853e+02
p_res: 2.141314e-14
d_res: 9.415485e-17
c_res: 1.124999e-11
number of iter: 21

--- my_pdipm ---
Elapsed time is 0.133457 seconds.
my_pdipm obj: 1.850853e+02
p_res: 2.128704e-14
d_res: 9.271789e-17
c_res: 1.107003e-11
number of iter: 23

=====
Trial 2: (m,n) = (1000,5000)
--- Matlab linprog ---
Elapsed time is 4.404320 seconds.
linprog obj: 3.703646e+02
p_res: 6.355627e-09
d_res: 6.070283e-13
c_res: 7.875680e-10
number of iter: 15

--- yz_pdipm ---
Elapsed time is 0.847172 seconds.
yz_pdipm obj: 3.703646e+02
p_res: 1.621035e-11
d_res: 1.095792e-16
c_res: 5.392526e-12
number of iter: 23

--- my_pdipm ---
Elapsed time is 0.847153 seconds.
my_pdipm obj: 3.703646e+02
p_res: 1.544362e-11
d_res: 1.089635e-16
c_res: 5.310377e-12
number of iter: 25
```

Figure 11: Matlab Screen Printout from file test_ipm1.m with $r = 4$


```
=====
Trial 3: (m,n) = (1500,7500)
  --- Matlab linprog ---
Elapsed time is 18.729498 seconds.
linprog obj: 5.519924e+02
p_res: 9.823734e-11
d_res: 3.568438e-11
c_res: 2.498966e-11
number of iter: 16

  --- yz_pdipm ---
Elapsed time is 3.379914 seconds.
yz_pdipm obj: 5.519924e+02
p_res: 3.420276e-12
d_res: 1.175165e-16
c_res: 2.824431e-10
number of iter: 25

  --- my_pdipm ---
Elapsed time is 3.379814 seconds.
my_pdipm obj: 5.519924e+02
p_res: 3.349875e-12
d_res: 1.118341e-16
c_res: 2.724191e-10
number of iter: 24

=====
Trial 4: (m,n) = (2000,10000)
  --- Matlab linprog ---
Elapsed time is 36.822342 seconds.
linprog obj: 7.507251e+02
p_res: 5.005713e-10
d_res: 7.161362e-16
c_res: 7.966327e-10
number of iter: 15

  --- yz_pdipm ---
Elapsed time is 6.275263 seconds.
yz_pdipm obj: 7.507251e+02
p_res: 1.562911e-11
d_res: 1.275385e-16
c_res: 3.238658e-10
number of iter: 24

  --- my_pdipm ---
Elapsed time is 6.275251 seconds.
my_pdipm obj: 7.507252e+02
p_res: 1.488896e-11
d_res: 1.256792e-16
c_res: 3.114017e-10
number of iter: 25
```

Figure 12: Matlab Screen Printout from file test_ipm1.m with $r = 4$

```
===== Statistics =====  
Solvers: linprog yzpdipm mypdipm  
Average iter: 15  
Average iter: 23  
Average iter: 24  
Average time: 15.064366  
Average time: 2.658919  
Average time: 2.622271  
Average resi: 2.166161e-09  
Average resi: 1.645583e-10  
Average resi: 1.584763e-10  
=====
```


 >>

Figure 13: Matlab Screen Printout from file test_ipm1.m with $r = 4$

MATLAB Screen printout for the run results of test_ipm2.m

```

Command Window

>> test_ipm2
--- Image cameraman: 40.00% pixels missing
--- Initial SNR = 6.12327

Recovery by linprog ...
number of iterations: 12
Elapsed time is 1.130550 seconds.
Recovered SNR = 23.6039

Recovery by yzpdipm ...
iter 1: [primal dual gap] = [3.41e+00 6.37e+02 1.49e+08]
iter 2: [primal dual gap] = [1.46e-15 4.36e+01 1.10e+00]
iter 3: [primal dual gap] = [1.38e-14 2.17e-02 3.19e+01]
iter 4: [primal dual gap] = [3.27e-15 9.47e-17 1.04e+01]
iter 5: [primal dual gap] = [6.12e-15 1.02e-16 8.40e+00]
iter 6: [primal dual gap] = [2.58e-15 1.02e-16 6.90e+00]
iter 7: [primal dual gap] = [2.28e-15 1.00e-16 7.59e+00]
iter 8: [primal dual gap] = [3.66e-15 9.89e-17 6.74e+00]
iter 9: [primal dual gap] = [8.83e-15 1.08e-16 8.61e-01]
iter 10: [primal dual gap] = [1.87e-14 1.17e-16 2.41e-01]
iter 11: [primal dual gap] = [3.32e-14 1.20e-16 7.65e-02]
iter 12: [primal dual gap] = [5.49e-14 1.24e-16 2.01e-02]
iter 13: [primal dual gap] = [8.30e-14 1.30e-16 1.92e-03]
iter 14: [primal dual gap] = [1.01e-13 1.34e-16 3.01e-06]
iter 15: [primal dual gap] = [1.02e-13 1.39e-16 2.63e-10]
number of iterations: 15
Elapsed time is 11.375359 seconds.
Recovered SNR = 23.6461

Recovery by mypdipm ...
iter 1: [primal dual gap] = [ 1.87e+02 2.81e+04 3.63e+11]
iter 2: [primal dual gap] = [ 9.36e-02 2.00e+03 2.26e-01]
iter 3: [primal dual gap] = [ 4.68e-05 9.98e-01 1.99e-02]
iter 4: [primal dual gap] = [ 2.34e-08 4.99e-04 1.30e-01]
iter 5: [primal dual gap] = [ 7.06e-10 2.50e-07 1.64e-02]
iter 6: [primal dual gap] = [ 3.92e-10 2.30e-07 1.13e-02]
iter 7: [primal dual gap] = [ 1.93e-10 1.99e-07 9.69e-03]
iter 8: [primal dual gap] = [ 7.43e-11 1.58e-07 9.08e-03]
iter 9: [primal dual gap] = [ 2.37e-11 1.17e-07 8.52e-03]
iter 10: [primal dual gap] = [ 6.92e-12 6.69e-08 9.76e-03]
iter 11: [primal dual gap] = [ 1.74e-12 3.15e-08 2.47e-02]
iter 12: [primal dual gap] = [ 3.63e-13 7.43e-09 3.37e-03]
iter 13: [primal dual gap] = [ 7.09e-14 1.40e-09 6.84e-04]
iter 14: [primal dual gap] = [ 3.39e-14 3.16e-10 2.21e-04]
iter 15: [primal dual gap] = [ 5.14e-14 8.20e-11 6.97e-05]
iter 16: [primal dual gap] = [ 8.41e-14 1.20e-11 1.41e-05]
number of iterations: 16
Elapsed time is 10.835465 seconds.
Recovered SNR = 23.6108

fx >>

```

Figure 14: Matlab Screen Printout from file test_ipm2.m

MATLAB Output Figures

Left: Original. Middle: Contaminated (SNR: 6.12). Right: Recovered (SNR: 23.6)

**Recovery by linprog**

Left: Original. Middle: Contaminated (SNR: 6.12). Right: Recovered (SNR: 23.6)

**Recovery by yzpdipm**

Left: Original. Middle: Contaminated (SNR: 6.12). Right: Recovered (SNR: 23.6)

**Recovery by mypdipm**

Figure 15: Matlab Output figures from file test_ipm2.m

Short Summary

Introduction This project aims to apply primal-dual interior point method to solve linear programming. The basic idea is to solve the primal and dual simultaneously, i.e., keep the primal feasibility and dual feasibility, and try to achieve the complementary condition with iteration going on.

Performance summary This algorithm has fast convergence rate in general. As shown in the MATLAB screen printout, the gap between primal and dual optimal solution decreases with order $O(n^{3.5})$.

Care about initial guess Do care about initial guess of feasible solution of LP, otherwise with the iteration going on, you may find the computer cannot compute the Cholesky factorization. One relatively good initial guess is $100 \times \max\{A_i\} * \mathbf{1}$ for the specific LP satisfying the equality constraint $(\mathbf{A}\mathbf{x})_i = \sum_j A_{ij}$

Save the necessity before computation For example, we need to use the values for $-x.*rd+rc$ for many times, so we can save it in advance and call them if necessary; moreover, saving the matrix $\text{diag}(x_1/z_1, \dots, x_n/z_n)$ is meaningless, since we can arrange computation such that it suffices to save the vector form $x./z$ instead.

Appreciate the sparse form and Cholesky factorization The Cholesky factorization gives us an alternative to obtain an inverse of a large sparse matrix \mathbf{A} , otherwise in each iteration we need to compute the matrix inverse inefficiently. Also, if order the columns of \mathbf{A} appropriately by using command $p = \text{colamd}(\mathbf{A})$ and $\text{chol}(\mathbf{A}(p,p))$, the computation will be much more easier.

Appreciate Predictor-Corrector Algorithm

- **Introduction:** The prediction step aims to *extrapolate* the optimal value of the next iteration, i.e., calculate an initial guess of the best output for next iteration; the corrector step aims to improve the initial guess using some rules such as *trapezoidal rule*. Such an algorithm is useful in solving differential equation, but recently we find that it is also useful for optimization problem.
- **Advantage:** By applying this method, we can solve the linear system in a more efficient and accurate way than the old algorithm. In particular, the disadvantage for the older one is that sometimes the linear system is singular, which makes the computer unable to continue the computation. However, this algorithm can solve the linear system correctly, which crosses over this obstacle.