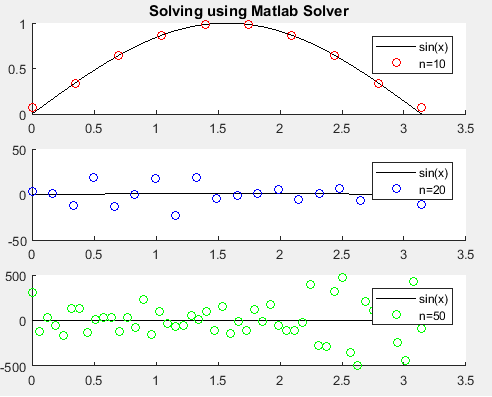
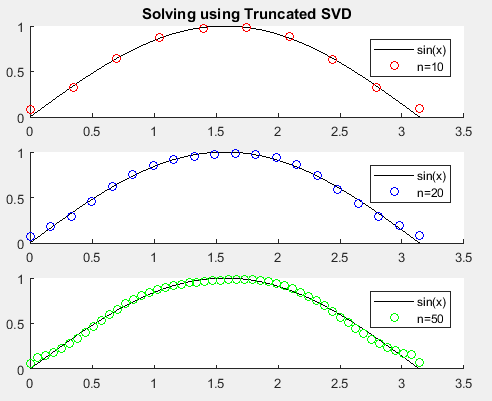
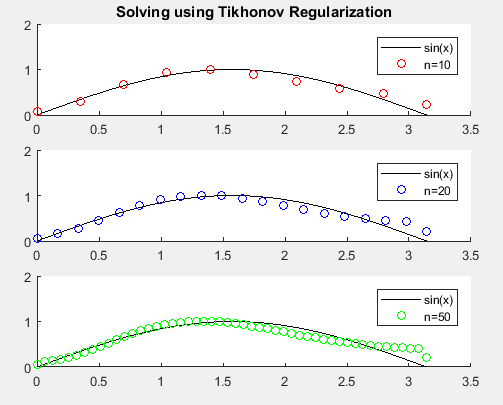
Numerical Computing Homework 5

Samuel Vidovich

Some Visuals:





Questions answered

1. Using A\b works great when n is small. After that, it is (technically speaking) a heap of garbage. As you can see, it explodes and fails to represent the true solution at all.
2. This one is my favorite. No matter how big n gets, it stands firmly on the true solution.
3. This one is pretty good – it starts to diverge a little bit the further away from origin we get!

The Code

% This is the main part of the program. It solves the integral equation

% using various methods and step sizes, and then plots the solutions in

% comparison to the true solution!

% This is less elegant than it could be, though still functional.

[A10, b10, s10, t10] = generateEquations(10);

[A20, b20, s20, t20] = generateEquations(20);

[A50, b50, s50, t50] = generateEquations(50);

% Matlab Built-in

x10 = A10\b10;

x20 = A20\b20;

x50 = A50\b50;

figure(1);

subplot(3,1,1)

hold on

plot(t50, sin(t50),'k');

plot(t10, x10,'ro');

hold off

legend('sin(x)','n=10');

title('Solving using Matlab Solver');

subplot(3,1,2)

hold on

plot(t50, sin(t50),'k');

plot(t20, x20,'bo');

hold off

legend('sin(x)','n=20');

subplot(3,1,3)

hold on

plot(t50, sin(t50),'k');

plot(t50, x50,'go');

hold off

legend('sin(x)','n=50')

% Truncated SVD

x10 = truncatedSVD(A10, b10);

x20 = truncatedSVD(A20, b20);

x50 = truncatedSVD(A50, b50);

figure(2)

subplot(3,1,1)

hold on

plot(t50, sin(t50),'k');

plot(t10, x10,'ro');

hold off

legend('sin(x)','n=10');

title('Solving using Truncated SVD');

subplot(3,1,2)

hold on

plot(t50, sin(t50),'k');

plot(t20, x20,'bo');

hold off

legend('sin(x)','n=20');

subplot(3,1,3)

hold on

plot(t50, sin(t50),'k');

plot(t50, x50,'go');

hold off

legend('sin(x)','n=50')

% Tikhonov regularization

x10 = tikhonov(A10, b10);

x20 = tikhonov(A20, b20);

x50 = tikhonov(A50, b50);

figure(3)

subplot(3,1,1)

hold on

plot(t50, sin(t50),'k');

plot(t10, x10,'ro');

hold off

legend('sin(x)','n=10');

title('Solving using Tikhonov Regularization');

subplot(3,1,2)

hold on

plot(t50, sin(t50),'k');

plot(t20, x20,'bo');

hold off

legend('sin(x)','n=20');

subplot(3,1,3)

hold on

plot(t50, sin(t50),'k');

plot(t50, x50,'go');

hold off

legend('sin(x)','n=50')

pause;

close all;

% This function uses a trapezoid rule approximation to generate the

% Matrix A, and RHS function values to generate b.

% The integral equation these represent is

% int(0,pi)[exp(s\*cos(t))\*x(t)dt] = (exp(s)-exp(-s))/s

%

% The function takes a value n -- think resolution, and returns:

% 1. The matrix A

% 2. The the vector b resulting

% 3. The points s(i), used to generate b

% 4. The points t(i), used to approximate x(t)

function [A, b, s, t] = generateEquations(n)

% Preallocate space for everything we need.

A = zeros(n,n);

b = zeros(n,1);

s = zeros(n,1);

t = zeros(n,1);

% Generate equally spaced points in [0, pi/2]

for i = 1:1:n

s(i) = (pi\*(i-1))/(2\*(n-1));

end

% Generate equally spaced points in [0, pi]

for i = 1:1:n

t(i) = (pi\*(i-1))/(n-1);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Now it's time to fill A. The formulas for this can be found in my %

% notes. %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Fill the first column.

for i = 1:1:n

A(i,1) = ((t(2)-t(1))/2)\*exp(s(i)\*cos(t(1)));

end

% Fill the last column

for i = 1:1:n

A(i,n) = ((t(n)-t(n-1))/2)\*exp(s(i)\*cos(t(n)));

end

% Fill the middle columns.

for i = 1:1:n % We will fill every row,

for j = 2:1:(n-1) % But we have already filled columns 1 and n.

A(i,j) = ((t(j+1)-t(j-1))/2)\*exp(s(i)\*cos(t(j)));

end

end

% Now let's generate b.

% The function as outlined in the description is not continuous at 0,

% and with the way the generation of s(i)'s are defined, s(1) is always

% 0. We have options, though: The limit of the function at 0 is 2. So

% instead of trying to generate b(1), we'll just let b(1) = 2.

% The rest of the vector can be generated normally.

b(1) = 2;

for i = 2:1:n

b(i) = ((exp(s(i)) - exp(-s(i))))/(s(i));

end

end

function x = truncatedSVD(A, b)

% A is nxn; get n. We'll need it for indexing shortly.

[n, ~] = size(A);

% Compute the SVDecomposition of A.

[U, S, V] = svd(A);

% Let's get U transpose, eh?

Ut = U';

% Initialize x. Who cares how big it is.

x = 0;

% Initialize a vector to hold our singular values.

% Since we don't know how many we're going to keep yet, this

% vector is going to increase in size.

sv = 0;

% Now let's choose only the singular values greater than 10^-6.

for i = 1:1:n

% S, being the matrix of singular values, is diagonal.

if S(i,i) >= 10^-6

% This should work: ... >= s(i-1) >= s(i) >= s(i+1) >= ...

sv(i) = S(i,i);

end

end

% Now, the length of sv is going to be the size of our truncated A.

% But really we're not at all interested in A, but its inverse.

% We're going to do this all in one loop. To hell with it!

k = length(sv);

for i = 1:1:k

x = x + ((1/sv(i))\*V(:,i)\*Ut(i,:)\*b);

end

end

function x = tikhonov(A, b)

% We will need these matrices!

[U, S, V] = svd(A);

% Do this to prevent syntax errors later.

Ut = U';

% A is nxn. Use n for indexing.

[n, ~] = size(A);

% Initialize x: We're going all the way to n unlike in the truncated

% SVD method, so it has definite size.

x = zeros(n,1);

% The penalty, as prescribed by Dr. Zheng.

m = 10^(-3);

% We will hold A's singular values in a vector for easy use.

sv = zeros(n,1);

% Stash the singular values:

for i = 1:1:n

sv(i) = S(i,i);

end

% Now let's compute x.

for i = 1:1:n

x = x + ((1/(sv(i) + (m/sv(i))))\*V(:,i)\*Ut(i,:)\*b);

end

end