CPSC 406, Term I, 2016/17
Assignment 1, due Thursday, September 22

Please show all your work: provide a hardcopy of the entire assignment (including plots and programs); in addition, e-mail your MATLAB programs to `cs406@ugrad.cs.ubc.ca`. When e-mailing your programs, include your name and student ID in the message's title.

Do not e-mail a complete assignment: only the programs.

Each of the first three questions is worth 25 marks, the fourth is worth 40 marks; with the maximum total not exceeding 100 marks.

1. (convexity)

   (a) Determine the values of α, β and γ (including negative and zero values) for which the function $f(x) = \alpha x^2 + \beta x + \gamma$ is convex, concave, both, or neither on the real line. Justify your answer.

   (b) Determine whether the following functions are convex, concave, both, or neither on the real line:

   i. $f(x) = -\sqrt{1+x^4}$

   ii. $f(x) = 5x - 6$

   iii. $f(x) = \frac{2x}{1+x^3}$

   (c) Determine whether

   $$f(x_1, x_2) = 2x_1^2 - 3x_1x_2 + 5x_2^2 + 17x_1 - 17x_2$$

   is convex, concave, both, or neither for $\mathbf{x} \in \mathbb{R}^2$.

2. (minimizing a rough function)

   Consider minimizing the scalar function

   $$f(x) = \frac{1}{2}(x-b)^2 + \lambda|x|,$$

   where b and λ are given real parameters, and $\lambda \geq 0$.

   (a) Show that this function is convex on the real line.

   (b) Find the unique minimizer $x^* = x^*(b; \lambda)$.

   [Note: this is a pen-and-paper question.]

3. (proving quadratic convergence of Newton's method)

   The handout on `Convergence of Newton's method`, available on the course home page, details a proof of quadratic convergence for Newton's method applied to systems of nonlinear equations.

   Prove a similar theorem for the case of just one equation in one unknown **without** using any integral or integration whatsoever.

4. (writing efficient code for solving block-tridiagonal linear systems)

   A *tridiagonal* $n \times n$ matrix A has only three potentially nonzero diagonals: the main one as well as the one below and the one above it. Solving a system of equations $A\mathbf{x} = \mathbf{b}$ for a given right hand side vector $\mathbf{b}$ with such a matrix, using an appropriate version of Gaussian elimination, requires only $\mathcal{O}(n)$ operations and only $\mathcal{O}(n)$ storage locations. Such an algorithm is particularly elegant if no pivoting is required. Here is a MATLAB function that carries this out:

```
function x = trid (md,ld,ud,b)
%
%  Solve Ax = b for a tridiagonal A
%  md = diagonal vecotor of A (length n)
%  ld = lower diagonal of A (length (n-1)
%  ud = upper diagonal of A (length (n-1)
%  b = right hand side vector

% LU decomposition (overwriting entries)
n = length(b);
for k=1:n-1
    ld(k)=ld(k)/md(k);              % compute multiplier
    md(k+1)=md(k+1)-ld(k)*ud(k);    % update pivot
    b(k+1)=b(k+1)-ld(k)*b(k);       % update r-h-s
end

% backward substitution
x=b;
x(n)=x(n)/md(n);
for k=n-1:-1:1
    x(k)=(x(k)-ud(k)*x(k+1))/md(k);
end
```

   See Chapter 5 of Ascher-Greif for the scoop on Gaussian elimination, LU decomposition and banded matrices.

   Your job in this exercise is to devise such an algorithm for a **block-tridiagonal** matrix and implement it in a function called, say, `block_trid`. Thus, each "element" of the $mn \times mn$ matrix A is now not a scalar but an $m \times m$ matrix (or *block*). If m is independent of n and $m \ll n$, which we shall assume (say, $m = 3$ and $n = 10,000$), then the matrix A is still tightly banded and its decomposition still requires only $\mathcal{O}(n)$ operations.

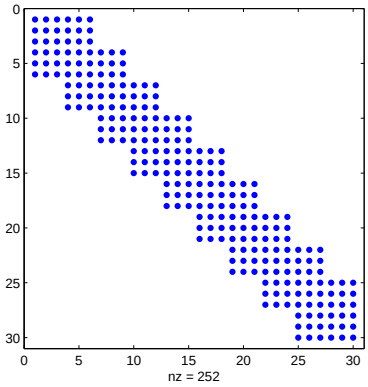
Assume that no partial pivoting is required. Your function `block_trid` should consume only $\mathcal{O}(n * m^2)$ storage locations in total and require at most only $\mathcal{O}(n * m^3)$ elementary operations (flops).

Test your function as follows: invent appropriate $m > 1$, $n \gg m$, a corresponding block diagonal matrix A and an exact solution $\mathbf{x}_{\text{true}}$. Then define $\mathbf{b} = A\mathbf{x}_{\text{true}}$, "forget $\mathbf{x}_{\text{true}}$ momentarily", and proceed to solve the system $A\mathbf{x} = \mathbf{b}$ using your `block_trid`. Following this, report $\|\mathbf{x} - \mathbf{x}_{\text{true}}\|$, which should be really small.

Example: The following script defines a block-tridiagonal matrix with $n = 10$ and $m = 3$, and displays its sparsity pattern (i.e., where the nonzeros are). [Note that since $nm = n * m$ is not very large we can afford to use a simple full $nm \times nm$ array for defining A. In your program, however, you don't have such luxury because n may be very large. So you must use a data structure that only requires $\mathcal{O}(n * m^2)$ storage locations.]

```
% example for Q4 asg 1, 2016

m = 3;
n = 10;
% define and print out block matrices (same for all k)
mblock = [1,-.1,-.1;-.1,1,-.1;-.1,-.1,1]
lblock = [.1,.5,.5;-.5,.1,.5;-.5,-.5,.1]
ublock = [.1,-.5,-.5;.5,.1,-.5;.5,.5,.1]

nm = n*m;                                % dimension of A
% Following is the sort of instruction to avoid for large n!!
A = zeros(nm,nm);

for k = 1:n
   A((k-1)*m+1:k*m,(k-1)*m+1:k*m) = mblock;
    if k > 1, A((k-1)*m+1:k*m, (k-2)*m+1:(k-1)*m) = lblock; end
    if k < n, A((k-1)*m+1:k*m, k*m+1:(k+1)*m) = ublock; end
end

spy(A) % display the sparsity pattern of A
```

This produces the plot


0
5
10
15
20
25
30
0
5
10
15
20
25
30
nz = 252