Consider the limited memory BFGS quasi-Newton method for unconstrained minimization of large scale problems:

\[
\min_{x \in \mathbb{R}^n} f(x_1, x_2, \ldots, x_n)
\]

The inverse Hessian approximation is obtained by applying \( m \) BFGS updates to the diagonal matrix \( H_0 \) using information from the previous \( m \) steps.

\[
H_{k+1} = (I - \rho_k s_k y_k^T) H_k ((I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T)
\]

\[
s_k = x_{k+1} - x_k
\]

\[
y_k = \nabla f_{k+1} - \nabla f_k
\]

\[
\rho_k = \frac{1}{y_k^T s_k}
\]

for the inverse Hessian approximation \( H_k \).

Here an approximation to the inverse Hessian \( H_{k+1} \) is obtained by applying \( m \) BFGS updates to the diagonal matrix \( H_0 \) using information from the previous \( m \) steps.

Please read the paper of Liu and Nocedal (1989).

Also read in class book pages 224-233.

Use the attached code lbfgs.tar, untar it by using command

tar – xvf and run the makefile using command make. This will create an executable sdrive.

Please modify the main driver routine sdrive or write equivalent code by yourself and test it for the difficult 4 variables Woods function:

\[
f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2
\]
\[
+10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1)
\]

a) Flowchart the code of lbfgs.f
b) Test the code using the starting point:

\[ x = (-3, -1, -3, -1)^T \]

c) Compare the performance of L-BFGS in terms of CPU time, number of iterations and gradient norm for the cases \( m=3, m=5 \) and \( m=7 \).

You should find that as \( m \) increases the required number of iterations for convergence decreases.

d) Compare the results with full BFGS by using routine conminf.f with imeth=1.