ANÁLISIS APLICADO
PROYECTO II

Condiciones para entregar el proyecto

1. Cada equipo debe tener de 3 a 4 miembros.
Por correo electrónico deben registrar el equipo (como en el proyecto 1). Cuando recibo este correo les asignaré (aleatoriamente) un conjunto de parámetros para la 2^a función (pág. 4).

2. Fecha de entrega: ver *comunidad.itam* → *Trabajos y Exámenes* → *Proyecto II*

3. Resolver las tareas abajo.

4. ¿Qué deben entregar como? Subir 2 objetos a *comunidad.itam*.

a) El código en un archivo comprimido (en formato `.zip`). Este debe incluir

- Todo lo que se requiere para correr el código debe ser contenido en el `zip`, *e.g.* si aproximan el Gradiente con nuestra función hecha en Lab. 1, entonces hay que incluir esa función. En particular: `lineSearch.m`, `pCauchy.m`, `rcSR1.m`, `lsBFGS.m` y `lsBFGSLiMem.m`.
- Para cada tabla de resultados en su documentación un *script* que reproduce ciertos datos. (No espero que tiempos coincidan.)
- Por si incluye una gráfica en su documentación, entonces también tiene que entregar un *script* que reproduce la gráfica desde el mismo punto de vista. Ver `help view`.

b) Un documento en formato `.pdf` que junta y comenta los resultados obtenidos.
No quiero código en el documento.

Dr. Andreas Wachtel
Semestre: Primavera 2020.

El proyecto

1. Escribir funciones en MatLab u Octave.

Parámetros centrales (para búsqueda en linea y región de confianza):

$$c_1 = 10^{-4}\,, \qquad c_2 = 0.99\,, \qquad tol = 10^{-5}\,, \qquad \eta = 0.1\,, \qquad \Delta_{max} = 1.25\,.$$

Los comentarios están en Ingles, pues MatLab no me acepta acentos.

1. Una función que calcula el punto de *Cauchy* (como en el proyecto 1):

```
function [pC] = pCauchy( B, g, delta )
```

2. Una función que implementa el método de región de confianza con actualizaciones simétricas de rango 1 (para B_k y H_k) y las siguientes restricciones:
   - Las actualizaciones de B_k, H_k, g_k solo se deben hacer cuando un paso es aceptado.
   - Además, las matrices solo se deben actualizar, bajo la condición
   $$|(\boldsymbol{\gamma} - B\boldsymbol{s})^T\boldsymbol{s}| \geq r\|\boldsymbol{s}\|_2\|\boldsymbol{\gamma} - B\boldsymbol{s}\|_2 \qquad \text{con} \qquad r = 10^{-6}\,.$$
   - solo se deben contar iteraciones en las cuales se actualiza x_k.
   - La signatura de la función debe ser:

```
function [x, iter] = rcSR1( f, x0, itmax )
% In :  f     ... (handle) function to be optimized
%       x0    ... (vector) initial point
%       itmax ... (natural number) upper bound for number of iterations
%
% Out:  x     ... (vector) last approximation of a stationary point
%       iter  ... (natural number) number of iterations
```

Ayuda: Recomiendo empezar con el pseudo código en Tema 4, Lab. 2, pero debido a mis restricciones arriba tienen que hacer cambios.

3. Una función que implementa el algoritmo de búsqueda en linea.
   La signatura de la función debe ser:

```
function [alpha, gnew] = lineSearch( f, xk, dk, gk )
% In :  f  ... objectve function (handle)
%       xk ... current point
%       dk ... chosen direction of descent
%       gf ... gradient of f in xk
%
% Out:  alpha ... a parameter satisfying (W1) and (W2)
%       gnew  ... gradient of f in  xk+alpha*dk
```

Ayuda: Se trata de implementar los algoritmos 3.5 y 3.6 en [Nocedal, ed. 2]. Para entender esos algoritmos les dí dos diagramas de flujo en Tema 2.2, ver clase 6.

4. Una función que implementa el método de BFGS con búsqueda en linea.
   Para encontrar el α_k deben usar su función `lineSearch.m`.
   La función debe tener los mismos argumentos y resultados que `rcSR1.m` y su signatura es:

```
function [x, iter] = lsBFGS( f, x0, itmax )
```

   Ayuda: Pueden usar el código que hicimos en clase (lo compartí con ustedes).

5. Una función que implementa el método de BFGS con memoria limitada y búsqueda en linea.
   Para encontrar el α_k deben usar su función `lineSearch.m`.
   La función debe tener los mismos resultados pero un argumento más que `rcSR1.m` y su signatura es:

```
function [x, iter] = lsBFGSLiMem( f, x0, itmax, m)
% In :  m ... number of recent update steps (limited memory parameter)
```

   Ayuda: Pueden usar el código que hicimos en clase (lo compartí con ustedes).

2. Tareas (aplicar su código).

2.1. Funciones. En *communidad.itam* encuentran el siguiente documento:

CUTE_UnconstrainedOptimizationTestFunctionsCollection.pdf

De este documento tomamos los puntos $\boldsymbol{x}_0$ iniciales que vienen con las funciones. Se deben implementar las siguientes dos funciones para hacer los experimentos:

- Rosenbrock (extended, p. 149) con un parámetro n.
- DIXMAANA (p. 155) con un parámetro n y para una letra (conjunto de parámetros). En el siguiente orden les asignaré los conjuntos de parámetros:

$$H, G, B, F, E, J, I, K, L$$

2.2. Simulaciones con la función de Rosenbrock.

Hay que aplicar los 3 métodos a la función para $n \in \{2, 8, 32, 128\}$. Para cada método haga una tabla que contiene los últimos 5 iteraciones con su número (para cada n), y los valores $\|\nabla f(\boldsymbol{x}_k)\|_2$, $f(\boldsymbol{x}_k)$, los errores $\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2$ y el tiempo.

¿En términos de tiempo, cual método conviene más?

¿En términos de iteraciones, cual método conviene más?

¿Puede relacionar el crecimiento del número de iteraciones (o del tiempo) con el de n ?

2.3. Simulación con la función DIXMAANA.

Solo aplique la función `lsBFGSLiMem.m` para $n \in \{240, 960\}$ y para $m \in \{1, 3, 5, 17, 29\}$.

Para cada dimensión n haga una tabla que contiene el número de iteraciones, los últimos valores $\|\nabla f(\boldsymbol{x}_k)\|_2$, $f(\boldsymbol{x}_k)$ y el tiempo.