ANÁLISIS APLICADO
PROYECTO I

Condiciones para entregar el proyecto

1. Cada equipo debe tener de 3 a 4 miembros.

Por correo electrónico deben registrar el equipo con los nombres de los miembros y el nombre de la función que van a optimizar. Si la función ya esta tomada de otro equipo, entonces les tengo que pedir que escogen otra. Las siguientes funciones se encuentran definidas en:

`https://www.sfu.ca/~ssurjano/optimization.html`

a) Himmelblau.
b) Styblinski-Tang
c) Booth
d) Beale
e) Matyas
f) Goldstein–Price function
g) McCormick
h) Easom
i) Three-Hump Camel
j) Lévi function N.13
k) Six-Hump Camel
l) GRIEWANK

2. Fecha de entrega: La fecha especificada en Canvas.

3. Entregar un documento en formato `.pdf` que junta y comenta los resultados de los problemas.
No necesito código en el documento.
Para cada gráfica deben entregar un código que la reproduce (sin cambios requeridos).
Para cada resultado deben entregar un código que lo reproduce.
Bajaré puntos si tengo que meter parámetros para reproducir sus resultados y gráficas.

4. Resolver los problemas abajo y entregar el código completo en un archivo comprimido de formato `.zip`. Este debe incluir todo lo necesario para reproducir las gráficas y resultados descritos en el PDF.

- pueden hacer sus plots con `plotly` o `matplotlib`.
- tienen que entregar también las funciones `.py` (o notebook) que usan para aproximar el gradiente y la Hessiana.

Primavera 2022, Dr. Andreas Wachtel.

El proyecto

1. Problemas: Escribir funciones en Python.

Parámetros centrales:

$$\eta = 0.1\ , \qquad tol = 10^{-5}\ , \qquad \Delta_{max} = 1.5\,.$$

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

- Una función que calcula el punto *dogleg*:

```
def pDogLeg( B, g, delta ):
# In :  B     ... an s.p.d. matrix that approximates the hessian of  f  in  xk
#       g     ... (vector) gradient of  f  in  xk
#       delta ... trust region radius
# Out:  p     ... The  dogleg  point
```

- Una función que calcula el punto de *Cauchy*:

```
def pCauchy( B, g, delta ):
# In :  B     ... (symmetric matrix) approximates the hessian of  f  in  xk
#       g     ... (vector) gradient of  f  in  xk
#       delta ... trust region radius
# Out:  pC    ... The  Cauchy  point
```

- Una función que implementa el siguiente método de región de confianza. Este método debe usar el punto de *Cauchy* y para el modelo cuadrático se usa $B_k \overset{\text{def}}{=\!=} \nabla^2 f(\vec{\boldsymbol{x}}_k)$.

```
def  mRC1( f, x0, itmax ):
# Trust region method using the  Cauchy  point.
#
# 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
#       msg ... (string) message that says whether (or not) a minimum was found
```

- Una función que implementa el siguiente método de región de confianza.
Este método debe usar el punto *dogleg*. Para asegurar que la hessiana B_k del modelo cuadrático siempre es s.p.d. la definimos de la siguiente manera (en cada paso): Usando `scipy.linalg.eigh` se determina solamente el eigenvalor más chico, *i.e.*, $\lambda_1 \overset{\text{def}}{=\!=} \lambda_{min}\big(\nabla^2 f(\vec{\boldsymbol{x}}_k)\big)$. Después, se usa

$$B_k \overset{\text{def}}{=\!=} \begin{cases} \nabla^2 f(\vec{\boldsymbol{x}}_k) & ,\ \text{si } \lambda_1 > 0 \\ \nabla^2 f(\vec{\boldsymbol{x}}_k) + sI & ,\ \text{si } \lambda_1 \le 0 \end{cases} \qquad \text{donde} \qquad s \overset{\text{def}}{=\!=} 10^{-12} - \frac{9}{8}\lambda_1\,.$$

```
def = mRC2( f, x0, itmax ):
# Trust region method using the  dogleg  point.
#   Same arguments and results as the  mRC1  method.
```

⋆ Para definir la respuesta `msg` se debe ver si el eigenvalor mínimo de $\nabla^2 f$ es no-negativo (? `scipy.linalg.eigh`).

2. Problemas: aplicar su código.

2.1. Dibujar en 3d. Sea $f\colon \mathbb{R}^2 \to \mathbb{R}$ y el punto inicial $\vec{\boldsymbol{x}}_0$ cerca de un mínimo local $\vec{\boldsymbol{x}}^\star$. La matriz Hessiana en $\vec{\boldsymbol{x}}_0$ sea simétrica y (semi-)definida positiva. Escoge una región de confianza con $\Delta > 0$. Dibuje la gráfica en $\mathbb{R}^3$ del modelo cuadrático en la región de la confianza.

Ayuda 1: ejemplo: plot en coordenadas polares `exampleCircular.m` en Canvas $\to$ `ayuda_P1_code.zip`
Ayuda 2: ejemplo para plots en python ver Canvas $\to$ `examples_plots.zip`
Ayuda 3: recomiendo `plotly`

2.2. Dibujar direcciones en 2d. Sea $f\colon \mathbb{R}^2 \to \mathbb{R}$ y el punto inicial $\vec{\boldsymbol{x}}_0$ cerca de un mínimo local $\vec{\boldsymbol{x}}^\star$. La matriz Hessiana en $\vec{\boldsymbol{x}}_0$ sea simétrica y (semi-)definida positiva y se usa para definir el modelo cuadratico en $\vec{\boldsymbol{x}}_0$. Escoge una región de confianza con $\Delta > 0$.

Luego haga un *plot* (en dos dimensiones) que contiene

- la frontera de la región de confianza
- algunos conjuntos de nivel en $\mathbb{R}^2$ del modelo cuadrático en la región de la confianza.
- los tres direcciones *Newton, Cauchy, dogleg.* Para obtenerlas use sus funciones `pDogLeg`, `pCauchy`.

Ayuda 1: ejemplo `example_contours` en Canvas $\to$ `ayuda_P1_code.zip`
Ayuda 2: se puede usar `matplotlib.pyplot.quiver`
Ayuda 3: La región de confianza se debe ver como un circulo: Sugiero ver:
`https://matplotlib.org/3.5.1/gallery/subplots_axes_and_figures/axis_equal_demo.html`
Ayuda 4: recomiendo `matplotlib`

2.3. Iteraciones y error. Para su función escoge un punto inicial tal que $\|\vec{\boldsymbol{x}}_0 - \vec{\boldsymbol{x}}^\star\|_2 > \Delta_{max}$ y en el cual la función tiene una Hessiana no positiva definida. En el caso, que la función es globalmente convexa, escoge una distancia mayor que $5\Delta_{max}$. Después, para cada método haga una tabla que contiene los últimos 8 iteraciones con su numero, y los valores $\|\nabla f(\vec{\boldsymbol{x}}_k)\|_2$ y $f(\vec{\boldsymbol{x}}_k)$.

Además, en el caso que la solución óptima es conocida mide los últimos 5 errores: $\|\vec{\boldsymbol{x}}_k - \vec{\boldsymbol{x}}^\star\|_2$.

2.4. Visualizar el proceso. Haga una gráfica similar a las en el Lab. 3 para visualizar el camino que tomó la sucesión $\{\vec{\boldsymbol{x}}_k\}$.

Ayuda 1: ejemplo `example_contours` en Canvas $\to$ `ayuda_P1_code.zip`
Ayuda 2: recomiendo `matplotlib`