Curso de Programación en C++.
TPL2. Trabajo Práctico de Laboratorio. [2019-10-23]

PASSWD PARA EL ZIP: 3QW6 GPCH F9CV

Ejercicios

[Ej. 1] **[surface]** Una superficie en el espacio puede representarse como una transformación de una región en el plano $(u, v) \in \mathbb{R}^2$ al espacio $(x, y, z) \in \mathbb{R}^3$. Por ejemplo la esfera puede escribirse como el mapeo de las coordenadas $u = \phi$ (coordenada *azimutal* o *longitudinal*), $v = \theta$ llamada *elevación* o *latitud*,

$$\begin{aligned} x &= R\cos(u)\ \cos(v), \\ y &= R\cos(u)\ \sin(v), \\ z &= R\sin(u). \end{aligned} \tag{1}$$

Esta transformación mapea el rectángulo $-\pi \leq u \leq \pi$, $-\pi/2 \leq v \leq \pi/$. Otras ejemplos de superficies son,

- **Elipsoide** de ejes a, b, c:

$$\begin{aligned} x &= a\cos(u)\ \cos(v), \\ y &= b\cos(u)\ \sin(v), \\ z &= c\sin(u). \end{aligned} \tag{2}$$

- **Cono** con eje alineado con el eje z y apertura angular θ. Usamos coordenadas polares en el plano: $u = \rho$ y $v = \phi$

$$\begin{aligned} x &= u\ \cos(v), \\ y &= u\ \sin(v), \\ z &= \frac{u}{\tan(\theta)}. \end{aligned} \tag{3}$$

El objetivo del ejercicio es escribir una clase virtual pura **`surface_t`** que representa una superficie genérica y una función **`area(surface_t &surf...)`** que calcula el área en una región de la superficie.

```
class surface_t {
  public:
  virtual void generate(VectorXd &uv,VectorXd &xyz)=0;
};

double area(surface_t &surf,int N=100,
double umin=0.0,double umax=1.0,
double vmin=0.0,double vmax=1.0);
```

- La clase **`surface_t`** tiene un método **`void generate(VectorXd &uv,VectorXd &xyz);`** que es el que representa el mapeo entre las coordenadas bidimensionales **`uv`** y las tridimensionales **`xyz`**.

- La función **`area()`** debe calcular la superficie mapeada por el rectángulo **`umin<=u<=umax`**, **`vmin<=v<=vmax`** en el plano **`(u,v)`** en la superficie 3D. Para esto debe dividir el rectángulo en **`N*N`** rectángulos, mapear los vértices de cada uno de estos rectángulos usando el método **`generate()`** y acumula la superficie en 3D. Recordar que para calcular el área de un triángulo en 3D definido por los 3 vértices x_0, x_1, x_2 conviene calcular las aristas a, b y utilizar la norma del producto vectorial

$$a = x_1 - x_0, \quad b = x_2 - x_0, \quad A = \tfrac{1}{2}\|b \times a\| \tag{4}$$

Nota1: Recordar que Eigen sólo acepta el producto vectorial **`a.cross(b)`** cuando **`a,b`** son **`Vector3d`** es decir de longitud fija. Si hacen los cálculos con vectores dinámicos entonces deben copiar estos en temporarios y utilizar el producto vectorial sobre éstos. La otra posibilidad es calcular el producto vectorial en forma explícita con las componentes de los **`VectorXd`**.
Nota2: Se provee el código fuente **`eigen6.cpp`** desarrollado en clase donde se calcula el área de un elipsoide como base para el ejercicio.

Verificaciones:

- **Plano:** Generado como $x = u, y = v, z = c + au + bv$. El área debe dar $\sqrt{1 + a^2 + b^2} \cdot A_{uv}$, donde A_{uv} es el área del rectángulo en el plano u, v: $A_{uv} = (u_{\max} - u_{\min})(v_{\max} - v_{\min})$.
- **Esfera:** Debe dar $4\pi R^2$.
- **Elipsoide:** Para un elipsoide oblongo con $a = b = 1, c = 2$ debe dar $A = 21.4784$. Para un elipsoide oblato con $a = b = 1, c = 0.5$ debe dar $A = 8.67185$. La fórmula analítica para el área se puede consultar aquí:
  https://en.wikipedia.org/wiki/Ellipsoid
- **Cono:** El área lateral del cono es $\pi R^2 / \sin(\theta)$.
- **Catenoide:** Es una figura de revolución alrededor del eje z definida así

$$x = a\cosh(v/c)\,\cos(u), \quad y = a\cosh(v/c)\,\sin(u), \quad z = v. \tag{5}$$

  Esta figura es muy interesante ya que representa la figura de área mínima cuando $a = c$ (en realidad es un **extremo** (**máximo** o **mínimo** local)). Calcular el área de la catenoide entre dos círculos de radio $R = 1$ a una distancia de $2L$ con $L = 0.6$ y para varios valores de c que controla la curvatura de la catenoide. Si variamos c entre 0.1 y 2 manteniendo R =cnst (es decir $a = R/\cosh(L/c)$) entonces tenemos los siguientes valores,

c	a(neck rat)	Area/Area(cilindro)
0.100	0.0049570	0.8624530
0.195	0.0920060	0.9159020
0.290	0.2486650	0.9507090
0.385	0.4030730	0.9548920
0.480	0.5295420	0.9452070
0.575	0.6267030	0.9351750
0.670	0.7000330	0.9295590
0.765	0.7554770	0.9284290
0.860	0.7978250	0.9304480
0.955	0.8306050	0.9342700
1.050	0.8563430	0.9389260
1.145	0.8768330	0.9438070
1.240	0.8933620	0.9485730
1.335	0.9068580	0.9530470

```
1.430   0.9180020   0.9571540
1.525   0.9272980   0.9608760
1.620   0.9351260   0.9642250
1.715   0.9417740   0.9672260
1.810   0.9474650   0.9699110
1.905   0.9523710   0.9723130
2.000   0.9566280   0.9744640
```

- Notar que el área está normalizada por el área correspondiente del cilindro $4\pi RL$.
- El área tiene extremos locales para $c = 0.385$ y $c = 0.765$ (un máximo y un mínimo, respectivamente).
- El valor de a en este caso coincide con el **neck ratio** es decir la relación entre el radio mínimo en el angostamiento ($z = 0$) y los extremos ($z = \pm L$).
- Además el área decrece uniformemente para $c \to 0$ ya que en ese caso el área tiende a la de dos discos en $z = \pm L$ es decir $A = 4\pi$.

[Ej. 2] **[optimize]** Escribir una clase **`double optimize(objfun_t &F,VectorXd &X);`** que toma una función objetivo **`F`** y busca el mínimo **`X`**, es decir tal que **`F(Y)>=F(X)`** para todo **`Y`**. La función objetivo está definida a través de una clase virtual pura

```
class objfun_t {
  public:
  virtual double eval(VectorXd &X)=0;
};
```

El algoritmo de optimización es muy simple, basado en gradiente. Dado un $X \in \mathbb{R}^N$ calcular el gradiente $g \in \mathbb{R}^N$ de F en X por diferencias finitas centradas

$$g_j = \frac{F(X + \epsilon e_j) - F(X - \epsilon e_j)}{2\epsilon} \tag{6}$$

donde ϵ es un parámetro a elegir por el usuario y e_j es el versor en la dirección j, es decir ($e_{jk} = \delta_{jk}$. El algoritmo consiste en avanzar una cierta distancia en la la dirección del gradiente con relajación ω, $X' = X - \omega g$. Esto se realiza iterativemente hasta que el gradiente del funcional g esté por debajo de una tolerancia especificada $\|g\| < \mathrm{tol}$.

Nota: El algoritmo puede ser divergente para ω grande, por lo tanto se recomienda probar con un ω pequeño (por ejemplo **`1e-3`**) e incrementarlo mientras el algoritmo sea estable.

Consigna: Escribir una función
`double optimize(objfun_t &F,VectorXd &X,double epsln=1e-3,double tol=1e-3);` que realiza la tarea indicada.

Verificaciones: Buscar el mínimo de los funcionales indicados a continuación

- Dado un **`X0`** de dimensión **`n`** podemos formar el funcional cuadrático

$$F = c + a\|X - X_0\|^2 \tag{7}$$

El mínimo se obtiene por supuesto en $X = X_0$ y el valor mínimo del funcional es c.
Probar en 3D con $X0 = (1, 1, 1)$, $a = 1$, $c = 2$ y otras variantes. **Consigna:**

- **Ley de refracción:** Queremos encontrar el tiempo mínimo para que una partícula recorra la distancia entre dos posiciones A y B (ver figura). La zona entre estos puntos está dividida en N bandas de ancho dado d_j ($N = 3$ en la figura) donde en cada banda la velocidad es constante e igual a c_j. Encontrar el recorrido óptimo, es decir las distancias L_0, L_1, L_2 tales que el tiempo sea mínimo. La distancia en la dirección paralela a las bandas $L = x_b - x_a$ es un dato.
  Ayuda: EL funcional a minimizar en este caso es el tiempo total de recorrido

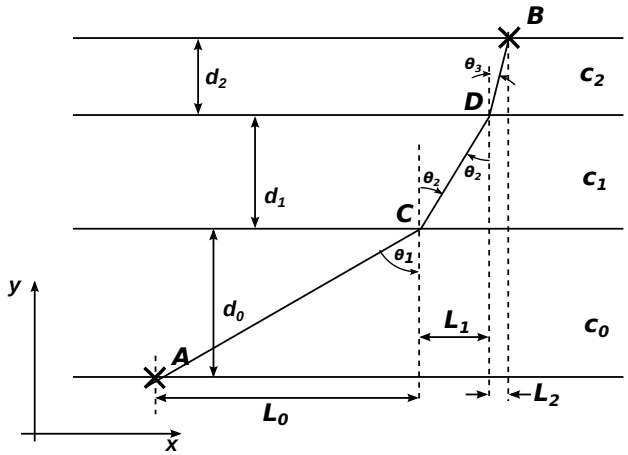
$$F = \sum_j \frac{(\text{distancia})_j}{(\text{velocidad})_j} = \sum_j \frac{\sqrt{d_j^2 + L_j^2}}{c_j} \tag{8}$$

  Las incógnitas son un vector de longitud $N - 1$,

$$X = (L_0, L_1, \ldots, L_{N-2}) \tag{9}$$

  ya que la longitud en la última capa está determinada por la suma, es decir

$$L_{N-1} = L - \sum_{j=0}^{N-2} L_j. \tag{10}$$

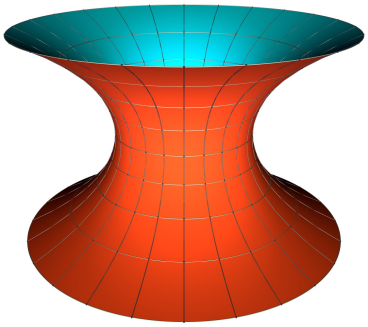


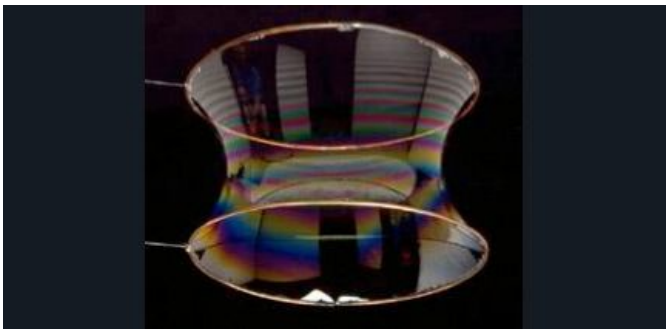
  Puede demostrarse que la condición que debe verificarse para que la solución sea un extremo es que se cumplan las **leyes de refracción de Snell**,

$$\frac{\sin(\theta_j)}{c_j} = \frac{\sin(\theta_{j+1})}{c_{j+1}} \tag{11}$$

  Consigna: Comprobar que, por ejemplo, si tenemos dos capas y $d_1 = d_2 = 1$, y $c_1 = 1, c_2 = 2$, $L = 1$. La solución es $L_1 = 0.29947$ $L_2 = 0.70053$.
- **Superficie mínima:** Dados dos círculos de radio R ubicados a una distancia $2L$ encontrar la superficie de área mínima que los vincula.
  Nota: Las superficies mínimas se pueden visualizar físicamente usando pompas de jabón, ya que la energía de una configuración está dada por su área y como el sistema busca la configuración de energía mínima, resulta ser que las pompas de jabón toman la forma que da área mínima. En el caso de dos círculos paralelos como se describe en este ejemplo la superficie de área mínima resulta ser la catenoide. Para más información ver
  - Catenoide: https://youtu.be/UuS2qcpeJ6c

- Superficies mínimas: https://youtu.be/dAyDi1aa40E, https://youtu.be/NscxxwDqq5s





Para representar la superficie vamos a utilizar una expresión de tipo superficie de revolución pero con una expansión en cosenos

$$\begin{aligned} x &= \rho(v)\ \cos(u), \\ y &= \rho(v)\ \sin(u), \\ z &= v, \end{aligned} \tag{12}$$

con

$$\begin{aligned} \rho(v) &= R(1 - a\cos(\zeta) - b\cos(3\zeta) - c\cos(5\zeta)\dots) \\ \zeta &= \pi v/2L, \end{aligned} \tag{13}$$

Los valores incógnitas para optimizar son $X = (a, b, c...)$

Consigna: Para $R = 1.0, L = 0.6$ y $m = 1$ términos en el desarrollo la superficie mínima se obtiene con los parámetros $X = (0.262601)$ y la superficie mínima es $A = 7.00032$. Con $m = 2$ parámetros obtenemos una superficie de $A = 6.99125$ y los parámetros óptimos son $X = (0.266887, -0.0115996)$.