Examen 02 - Programación e introducción a los métodos numéricos

William Urgent, woquendo@gmail.com

Abstract

Solamente se califica lo enviado al repositorio asignado hasta la hora límite. Debe estar presente en el salón para presentar el examen .

A menos que se indique lo contrario, **ningún** programa deberá pedir entrada de usuario. Haga uso de lo visto hasta ahora en el curso, además de lo que se pide explícitamente en cada punto. Si un programa no compila y/o no se ejecuta correctamente (error de sanitizer) se evaluará sobre un cuarto del valor del punto. **Debe hacer uso de funciones**

Importante Para facilitar la presentación del examen, el link que se le enviará le creará un repositorio con los programas base, es decir, **no debe usar `git init`**. Por favor, antes de continuar usted debe:

1. Aceptar la invitación del examen que se le ha enviado por correo.
2. En la consola, crear una carpeta nueva **POR FUERA** de cualquier repostiorio que ya tenga usted, y entrar a esa carpeta

```
mkdir NOMBRECARPETA
cd NOMBRECARPETA
```

3. Si está usando la red cableada, exportar el proxy

```
export https_proxy=...
```

   Recuerde que si ya ha ejecutado este comando puede acceder a él facilmente presionando `ctrl+r` y excribiendo las primeras letras `exp` .

4. Clonar el repositorio que ha sido creado para usted (la extrae en la dirección URL del navegador)

   ```
   git clone https://github.com/NOMBRECLASSROOM/EXAMNAME-NOMBREUSUARIO
   ```

   Esto ya dejará listo el repositorio con el remoto y demás. **No es necesario que configure el remoto**. Al haber clonado el remoto quedó configurado.

5. Entrar al repositorio clonado,

```
cd EXAMNAME-NOMBREUSUARIO
```

   e ir haciendo `git add`, `git commit` y `git push` para enviar el remoto.

1 (2.0/5.0) Matriz ortogonal

(Guarde la solución en el archivo `1.cpp`) Una matriz ortogonal es aquella que multiplicada por su transpuesta da la matriz idéntica (en otras palabras, aquélla cuya transpuesta es igual a su inversa):

$$A \cdot A^T = A^T \cdot A = 1 \quad . \tag{1}$$

Por ejemplo, la matriz

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.8 & 0.3 & -0.52 \\ 0 & -0.6 & 0.4 & -0.69 \\ 0 & 0 & 0.86 & 0.5 \end{pmatrix} \tag{2}$$

es ortogonal con una precisión de $\simeq 1.2\%$. Escriba un programa que tenga una función que reciba una matriz bidimensional de tamaño $n \times n$ (modelada con un `std::vector<double>` **unidimensional**) , y retorne un booleano indicando si la matriz es ortogonal o no, bajo una precisión dada). El prototipo de la función que verifica a la matriz debe ser

```
1  bool is_orthogonal(const std::vector<double> & M, const double eps)
```

donde eps denota la precisión con la que se desea verificar si la matriz es ortogonal. Verifique su programa usando la matriz mostrada más arriba. NO pida entrada de usuario. Nombre este archivo como `1.cpp` . Puede hacer la verificación elemento por elemento: primero calcula la transpuesta (esto no es necesario pero le puede facilitar pensar el problema), luego hace la multiplicación y compara elemento a elemento con la matriz identidad. Cada elemento debe cumplir que su diferencia porcentual con el de la matriz identidad debe ser menor que eps. Recuerde que la multiplicación de matrices se define como

$$C_{ij} = \sum_k A_{ik} B_{kj}. \tag{3}$$

A continuación se muestra la función main que debe usar y su salida correspondiente. Usted debe implementar las funciones que están declaradas pero no definidas.

```
#include <iostream>
#include <vector>
#include <cmath>

bool is_orthogonal(const std::vector<double> &  M, const double eps);

int main(void)
{
  const int N = 4;
  std::vector<double> A = {1, 0, 0, 0,
                           0, 0.8, 0.3, -0.52,
                           0, -0.6, 0.4, -0.69,
                           0, 0, 0.86, 0.5};
  std::cout << is_orthogonal(A, 0.001) << std::endl;
  std::cout << is_orthogonal(A, 0.012) << std::endl;
  std::cout << is_orthogonal(A, 0.1) << std::endl;

  return 0;
}
bool is_orthogonal(const std::vector<double> &  M, const double eps)
{
  //implemente el código acá
  // calcule la transpuesta
  // haga la multiplicacion en otro arreglo
  // calcule la diferencia comparando con la identidad
  // retorne true o false de acuerdo a lo anterior
}
```

```
0
1
1
```

2 (2.0/5.0) Ángulo entre dos vectores N-dimensionales

(Guarde la solución en el archivo `2.cpp`). En este punto usted escribirá un programa que simula vectores N-dimensionales usando arreglos (use `std::vector<double>`). A partir de estos dos vectores usted puede calcular el producto punto entre ellos, y de allí el ángulo relativo entre los mismos, usando la definición convencional

$$\vec{a} \cdot \vec{b} = ab \cos\theta = \sum_i a_i b_i,$$

donde $a(b)$ es la norma del vector a (b) y a_i (b_i) cada una de sus componentes. Para obtener el ángulo θ, puede usar la función `acos`. La norma es definida de la forma usual $c = \sqrt{\sum c_i^2} = \sqrt{\vec{c} \cdot \vec{c}}$.

Su programa debe calcular correctamente el ángulo entre dos vectores dados.

El siguiente código muestra la estructura base de su programa. Usted debe implementar las funciones que se llaman ahí, pero no puede cambiar la función main (puede hacer cambios temporales por ejemplo para probar con vectores más pequeños y de valores conocidos, pero al final debe dejar la misma forma). Se muestran dos algoritmos de la función estándar para llenar los arreglos. Puede encontrar más información en iota y fill .

```cpp
#include <iostream>
#include <algorithm>
#include <numeric>
#include <vector>
#include <cmath>

double compute_dotproduct(const std::vector<double> & VA, const std::vector<double> & VB);
double compute_angle(const std::vector<double> & VA, const std::vector<double> & VB);
double compute_norm(const std::vector<double> & V);

int main(void)
{
  const int N = 100;

  std::vector<double> A(N), B(N);
  // fill vectors (you can use other ways to fill them just to test)
  std::fill(A.begin(), A.end(), 1); // Fill A with 1 on all places
  std::iota(B.begin(), B.end(), 0); // fill B as 0, 1, 2, 3, 4, ..., N-1
  // Compute the relative angle
  std::cout << compute_angle(A, B) << std::endl;
}
double compute_dotproduct(const std::vector<double> & VA, const std::vector<double> & VB)
{
// Implemente su solucion
}
double compute_angle(const std::vector<double> & VA, const std::vector<double> & VB)
{
// Implemente su solucion
}

double compute_norm(const std::vector<double> & V)
{
// Implemente su solucion
}
```

Salida del programa para el vector mostrado:

```
0.52794
```

3 (2.0/5.0) Estadística de un vector

(Guarde su solución en el archivo `3.cpp`) En esta parte usted calculará algunas estadísticas de un vector n-dimensional, como el promedio, la desviación estándar, el mínimo y el máximo. Haga uso de los nuevos bucles `for (auto ... : array)` que se vieron en clase y vienen del estándar c++11. Para el programa de ejemplo, la salida debe ser como se muestra más abajo.

```cpp
#include <iostream>
#include <vector>
#include <cmath>

double mean(const std::vector<double> & data);
double sigma(const std::vector<double> & data);
double min(const std::vector<double> & data);
double max(const std::vector<double> & data);

int main(int argc, char **argv)
{
  std::vector<double> data = {2.34, 3.76, 4.00, 1.98, -6.56, 9.09, -8.09};
  std::cout << mean(data) << std::endl;
  std::cout << sigma(data) << std::endl;
  std::cout << min(data) << std::endl;
  std::cout << max(data) << std::endl;

  return 0;
}
```

```cpp
double mean(const std::vector<double> & data)
{
  // implemente su solucion
}
double sigma(const std::vector<double> & data)
{
  // implemente su solucion
}

double min(const std::vector<double> & data)
{
  // implemente su solucion
}
double max(const std::vector<double> & data)
{
  // implemente su solucion
}
```

0.931429
6.11788
-8.09
9.09