Instituto Tecnológico de Costa Rica
Área de Ingeniería en Computadores
Profesor: Dr. Pablo Alvarado Moya
CE-3102 Análisis Numérico para Ingeniería
II Semestre 2018

Entrega: **19 de setiembre, 2018**

Tarea 4

En esta tarea usted debe implementar tres métodos de descomposición para resolver sistemas de ecuaciones lineales.

Utilice el código base colocado en la carpeta Documentos/Tareas/tarea4 en el tecDigital. Dicho código utiliza la herramienta de configuración CMake para manejar la estructura de directorios y aplicaciones de prueba, medición de desempeño y programa principal.

Revise en particular la estructura de la clase `anpi::Matrix`. Note que esta clase requiere el estándar C++11 como mínimo.

Para vectores utilice `std::vector<T>`.

1. Sobrecargue los operatores aritméticos para producto, que permitan multiplicar matrices con matrices, y matrices con vectores. Los métodos deben verificar que las dimensiones de las matrices y vectores sean compatibles. De no ser así debe producir una excepción `anpi::Exception` con el mensaje adecuado. El código base ya brinda el espacio para este punto en el archivo `include/Matrix.tpp`

   Revise los estándares de estilo de Sutter y Alexandrescu en cuanto a sobrecarga de operadores.

2. Agregue en el archivo `test/testMatrix.cpp` pruebas unitarias para verificar que los productos se realizan correctamente, y que si se pasan matrices y/o vectores de tamaños incompatibles se producen las excepciones correspondientes.

3. Implemente la descomposición **LU** de una matrix con los métodos de Doolittle y de Crout, con las interfaces:

```
template<typename T>
void luDoolitle(const anpi::Matrix<T>& A,
                anpi::Matrix<T>& LU,
                std::vector<size_t>& p);

template<typename T>
void luCrout(const anpi::Matrix<T>& A,
             anpi::Matrix<T>& LU,
             std::vector<size_t>& p);
```

   donde la matriz `LU` tendrá las dos matrices codificadas tal y como se explicó en la clase. Ambos métodos deben utilizar pivoteo, y el vector `p` deberá contener la permutación de índices de filas, es decir, `p[0]` indica el índice de fila de la matriz

original `A` que se usó como primera fila en la descomposición, `p[1]` indica qué fila de la matriz original quedó en la segunda fila en la descomposición, y así sucesivamente.

Por ejemplo, si la matriz `A` y el vector de permutación `p` están dados por

$$\mathbf{A} = \begin{bmatrix} -1 & -2 & 1 & 2 \\ 2 & 0 & 1 & 2 \\ -1 & -1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \qquad \underline{\mathbf{p}} = \begin{bmatrix} 1 \\ 0 \\ 3 \\ 2 \end{bmatrix}$$

entonces la descomposición resultante en realidad corresponde a la matriz

$$\mathbf{A} = \begin{bmatrix} 2 & 0 & 1 & 2 \\ -1 & -2 & 1 & 2 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & 0 & 1 \end{bmatrix}$$

4. Implemente las funciones

```
template<typename T>
void unpackDoolittle(const Matrix<T>& LU,
                     Matrix<T>& L,
                     Matrix<T>& U);

template<typename T>
void unpackCrout(const Matrix<T>& LU,
                 Matrix<T>& L,
                 Matrix<T>& U);
```

   cuya función es desempaquetar la representación compacta `LU`, separándola en las dos matrices `L` y `U` según corresponda a cada método. Estos métodos son utilizados en las pruebas unitarias.

   Ponga especial cuidado a la diferencia entre los métodos de Doolittle y Crout en cuanto a cuál de las matrices `L` o `U` contendrá una diagonal de '1'.

5. Verifique sus funciones de descomposición LU con las pruebas unitarias ya disponibles en el directorio `test`.

   Agregue las pruebas unitarias que considere pertinentes (por ejemplo, para probar las funciones "unpack*" realizadas en el punto anterior).

   Sus métodos deben pasar todas las pruebas.

6. Compare los dos algoritmos anteriores en cuanto a tiempo requerido en función del tamaño de la matriz `A` (tamaño $n \times n$). Debe programar para ello dentro del marco ya disponible en el directorio `benchmarks` el código necesario para medir tiempos. Utilice los otros archivos de evaluación de la suma de matrices como referencia.

7. Utilice una función

```
template<typename T>
inline void lu(const anpi::Matrix<T>& A,
               anpi::Matrix<T>& LU,
               std::vector<size_t>& p);
```

para llamar al método que resulte más rápido de los dos implementados en los puntos anteriores.

8. Implemente ahora el método de descomposición **QR** de una matrix con la interfaz:

```
template<typename T>
void qr(const anpi::Matrix<T>& A,
        anpi::Matrix<T>& Q,
        anpi::Matrix<T>& R);
```

Debe utilizar las transformaciones de Householder para este fin, tal y como se describió en la clase.

9. Para verificar que su función de descomposición QR sea correcta, escriba pruebas unitarias en el directio `test` que verifiquen:

   9.1. que `Q` sea una matriz ortogonal probando que $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$.

   9.2. que `R` sea una matriz triangular superior

   9.3. que $\mathbf{QR} = \mathbf{A}$

   Observe que errores numéricos conducen a que las identidades anteriores solo se pueden cumplir parcialmente, en un cierto rango de precisión.

   Su método debe pasar todas las pruebas.

10. Implemente ahora los métodos

```
template<typename T>
bool solveLU(const anpi::Matrix<T>& A,
             std::vector<T>& x,
             const std::vector<T>& b);

template<typename T>
bool solveQR(const anpi::Matrix<T>& A,
             std::vector<T>& x,
             const vector<T>& b);
```

que internamente hacen uso de las descomposiciones implementadas en los puntos anteriores para resolver un sistema de ecuaciones lineales

$$\mathbf{A}\underline{\mathbf{x}} = \underline{\mathbf{b}}$$

Para la descomposición LU usted utilizará su método más rápido; es decir, usted utilizará la función `lu<T>`.

Note que con el método LU con pivoteo, usted en realidad está haciendo la descomposición

$$\mathbf{PA} = \mathbf{LU}$$

donde $\mathbf{P}$ representa la permutación de las filas de la matriz $\mathbf{A}$ según el resultado del vector de permutación. Dicha matriz se obtiene como

$$\mathbf{P} = \begin{bmatrix} \underline{\mathbf{u}}_{p[0]}^T \\ \underline{\mathbf{u}}_{p[1]}^T \\ \vdots \\ \underline{\mathbf{u}}_{p[n-1]}^T \end{bmatrix}$$

donde $\underline{\mathbf{u}}_i$ es un vector *canónico* con todos sus elementos en cero, excepto el i-ésimo que tiene valor uno. En otras palabras, la matriz $\mathbf{P}$ se obtiene permutando las filas de la matriz identidad en la misma forma indicada por el vector $\underline{\mathbf{p}}$.

Con lo anterior usted puede replantear el problema para poder utilizar sus métodos de descomposición **LU** con pivoteo.

11. Implemente pruebas unitarias para los métodos del punto anterior.

12. Implemente un método de inversión de matrices utilizando la descomposición LU con la interfaz:

```
template<typename T>
void invert(const anpi::Matrix<T>& A, anpi::Matrix<T>& Ai);
```

13. Agregue pruebas unitarias para la inversión de matrices.

Todos los métodos deben producir excepciones en caso de que ocurra algún problema y deben indicar en la excepción exactamente qué fue el problema (ver `anpi::Exception`). Las pruebas unitarias deben verificar que estas excepciones se producen correctamente, forzando las situaciones indebidas para eso.

Modularice su código en los archivos correspondientes.

Esta tarea es grupal.

Entregables:

1. Código fuente bien documentado.

2. Archivo README con instrucciones de cómo compilar y correr.

3. Pruebas unitarias integradas a las pruebas del directorio `test`. Dichas pruebas deben utilizar las precisiones `float` y `double` en todos los casos.

4. Mediciones de tiempo integradas en el directorio `benchmarks`.