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

Fecha de entrega: **5 de octubre, 2018**

Proyecto 2

1 Introducción

Este proyecto integra varios temas del curso e incorpora un aspecto de programación numérica asociado a arquitecturas con instrucciones SIMD (*single instruction multiple data*) para optimizar la ejecución de estructuras algorítmicas básicas.

Este tipo de optimizaciones forma parte de todas las bibliotecas numéricas estándar modernas y este proyecto pretende dar un vistazo general a la temática hasta un nivel que le permita optimizar la descomposición LU realizada en la tarea 4.

En la segunda parte del proyecto se utilizará la descomposición LU optimizada para resolver un problema concreto de modelado eléctrico en la búsqueda de trayectorias de un robot.

2 Objetivos

2.1 Objetivo general

Optimizar código de solución de sistemas de ecuaciones lineales y aplicarlo en un problema de ingeniería.

2.2 Objetivos específicos

1. Conocer los requisitos en el manejo de memoria para utilizar instrucciones SIMD
2. Optimizar una operación básica entre matrices utilizando instrucciones SIMD
3. Optimizar la descomposición LU y su uso en la solución de sistemas de ecuaciones lineales.
4. Evaluar el desempeño de los algoritmos implementados
5. Aplicar el código optimizado en una aplicación de ingeniería.

3 Procedimiento

El proyecto tiene dos partes: la optimización del código y el problema de aplicación. Usted utilizará como base para el proyecto el código entregado para tal fin.

3.1 Optimización del código

La optimización del código se realizará en el proyecto utilizando un set de instrucciones SIMD, en particular SSEx/AVX de Intel. Las instrucciones SIMD permiten ejecutar en paralelo una operación aritmética o lógica con todos los datos almacenados en registros especiales disponibles para este tipo de instrucciones. Desde C/C++ se accede a este tipo de instrucciones a través de funciones básicas llamadas *intrinsics*. Cada tipo de microprocesador (Intel, ARM, etc.) ofrece sus tipos particulares de instrucciones *intrinsics*, que por tanto no son portables entre arquitecturas. Es decir, si usted optimiza el código para AVX de Intel, su optimización solo funcionará para procesadores que soporten AVX. Por ello, el código optimizado usualmente se controla por medio de macros del preprocesador, que seleccionan entre código sin optimizar portable o códigos optimizados para arquitecturas particulares.

1. Seleccione una arquitectura SIMD disponible para todos los miembros de su equipo de trabajo: SSE2, SSE3, AVX, etc. Puede utilizar éste sitio para revisar las distintas tecnologías disponibles, y qué tipo de instrucciones ofrecen. Con

   ```
   > grep flags /proc/cpuinfo
   ```

   usted puede en GNU/Linux verificar qué tipo de arquitectura tiene su procesador (busque por avx, avx2, sse, etc.).

2. Asegúrese de comprender bien cómo se utiliza el archivo `cmake/AnpiConfig.hpp.in` para generar `include/AnpiConfig.hpp`, y cómo se usan los macros allí definidos, para activar o desactivar secciones de código durante la configuración del código (ver más información aquí). Usted deberá seguir la misma estrategia para seleccionar entre código optimizado y sin optimizar para así comparar el desempeño de ambas versiones.

3. Analice la clase `anpi::Matrix` en los siguientes puntos:

   3.1. Analice cómo se reserva la memoria de la matriz y cómo se libera. Debe comprender el concepto de `allocator` y cómo se utiliza para asegurar alineamiento de la memoria. Estudie cómo el código selecciona cuántos bytes de alineamiento utilizar para cada tipo de set instrucciones (SSEx, AVX, AVX-512-F, etc.)

   3.2. La matriz soporta rellenado (*padding*), es decir, reserva no solo el número especificado de columnas, sino espacio adicional de modo tal que cada fila contenga un número de entradas múltiplo entero del número de datos que caben en los registros del set de instrucciones SIMD seleccionado (por ejemplo, 128 bits para SSE2, o 256 bits para AVX, es decir 16 bytes o 32 bytes respectivamente). Nótese que este relleno es invisible al usuario de la matriz en el sentido de que el usuario no lo puede acceder directamente. La figura 1 ilustra la distribución de memoria en el caso donde se reservan 5 columnas en una configuración donde se desean emplear registros de 128 bits.

   Analice en la matriz de qué modo se decide la cantidad de relleno a utilizar.

   Estudie de qué modo se determina cual es el tamaño total de una fila.

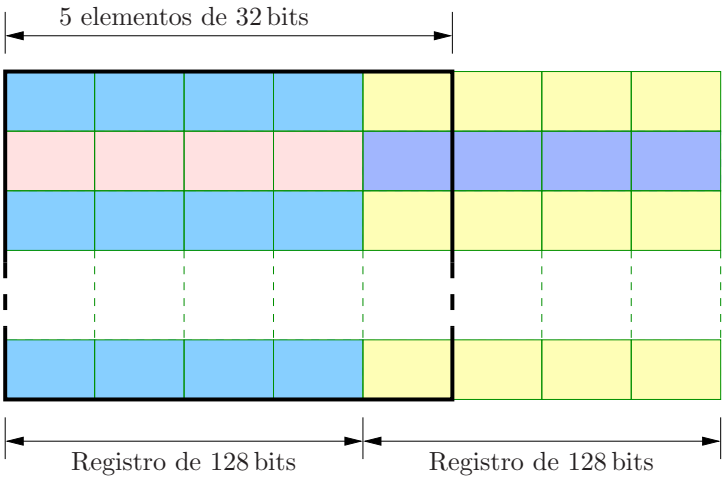

Figura 1: Ejemplo de distribución de memoria en una matriz con m filas y 5 columnas de 32 bits (por ejemplo, de tipo `int` o `float`) para usarla con registros de 128 bits.

3.3. Analice la implementación del `operator+` para `anpi::Matrix`. Este operador delega la implementación a funciones externas especializadas de tal modo que, en tiempo de compilación, se decide si existe una implementación optimizada para un tipo particular, o si debe utilizarse una función genérica para otros tipos.

4. Su primera tarea con SIMD consiste en acelerar las funciones `subtract` para realizar la resta de dos matrices. Su optimización debe ser hecha de tal modo que

   4.1. Solo se active cuando el usuario lo indique así en `AnpiConfig.hpp`. No debe agregar una bandera adicional a la ya existente.

   4.2. Debe ser acelerada para los tipos soportados por el set SIMD seleccionado anteriormente.

   4.3. Si un tipo no es soportado por SIMD, el método debe aun funcionar usando la implementación genérica.

5. Agregue pruebas unitarias a `testMatrix.cpp` para comprobar que el método `subtract` funcione correctamente con todos los tipos soportados.

6. Mida la diferencia de desempeño entre las implementaciones con y sin SIMD para el método `subtract`. Utilice como referencia a `benchmarkMatrixAdd.cpp`.

7. Ya que conoce los principios de optimización de código con vectorización manual, optimice ahora utilizando instrucciones SIMD el cálculo de la descomposición LU y de la función `solveLU`. Seleccione el algoritmo de descomposición visto en clase que sea más apto para la optimización SIMD.

8. Verifique con pruebas unitarias de la Tarea 4 que los métodos optimizados funcionen correctamente.

9. Mida el desempeño de su código optimizado contra el código sin optimizar para distintos tamaños de matrices, y grafique sus resultados. Realice las pruebas para la versión "Debug" y la version "Release" de su código (es decir, sin optimización y con optimización por parte del compilador).

3.2 Caso de aplicación

Como aplicación se utilizará la solución de sistemas de ecuaciones lineales en un problema que calcula la trayectoria a seguir por un robot. El robot recibe tanto un mapa que indica donde se encuentran obstáculos, tal y como lo ilustra la figura 2, así como las coordenadas de inicio y fin de la trayectoria en ese mapa.

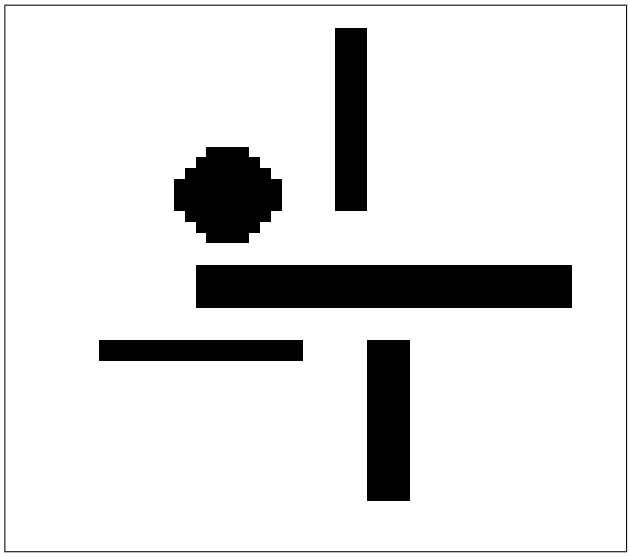

Figura 2: Mapa del área de movilización del robot. Las áreas blancas denotan paso libre y las negras obstáculos.

Para resolver este problema se utiliza una rejilla resistiva como la ilustrada en la figura 3. Los nodos en la rejilla corresponden cada uno a un pixel del mapa. El valor de resistencia es alto si uno o ambos terminales de la resistencia está pintado con negro en el mapa. El valor de las resistencias denota qué tan fácil le es al robot movilizarse por ese camino, así que se utilizarán resistencias en el rango de $1\,\Omega$ para el espacio libre y en el rango de $1\,\mathrm{M}\Omega$ cuando hay obstáculos.

En el punto inicial de la trayectoria se inyectará una corriente de $1\,\mathrm{A}$ y en el punto final se extraerá dicha corriente. En otras palabras, se conectará una fuente de corriente entre los nodos inicial y final de la trayectoria.

La idea de la aplicación es estimar todas las corrientes en la rejilla, para luego seguir dos estrategias para encontrar la trayectoria del robot. La primera estrategia es simple y sigue los valores de máxima corriente de salida en cada nodo. La segunda estrategia es más elaborada y simula la trayectoria de una partícula que se posiciona en un campo de fuerza.

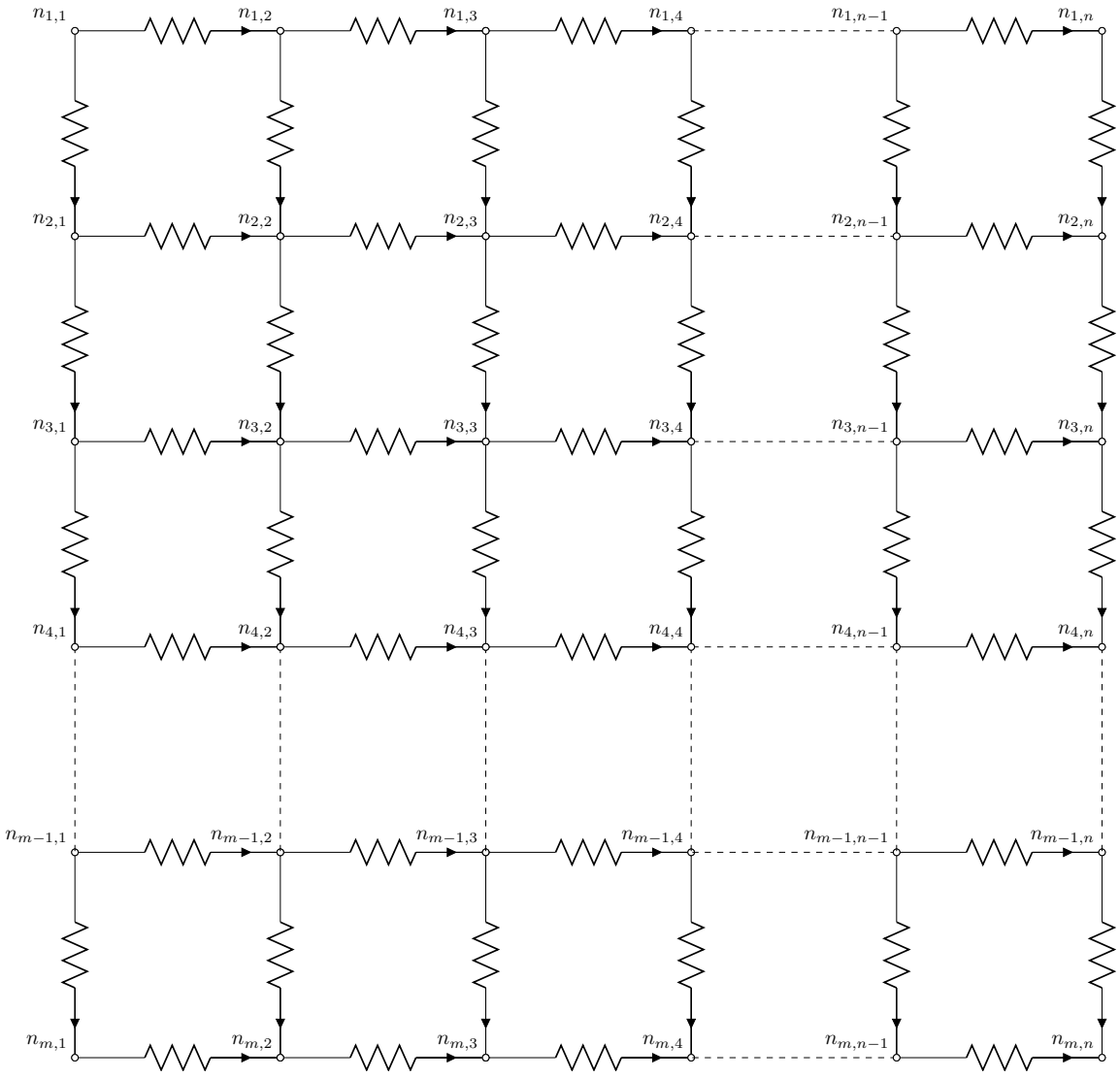

Figura 3: Malla resistiva para modelar búsqueda de trayectoria

La primera tarea consiste en plantear el sistema de ecuaciones $\mathbf{A}\underline{\mathbf{x}} = \underline{\mathbf{b}}$, donde el vector $\underline{\mathbf{x}}$ contiene todas las corrientes a ser calculadas.

Nótese que si la rejilla resistiva tiene tamaño $n \times m$, entonces tendremos $n-1$ resistencias en cada una de las m filas, y n resistencias en cada una de las $m-1$ bandas de resistencias verticales. Esto equivale a un total de $2nm-(m+n)$ resistencias, cuyas corrientes deben ser calculadas. Por ejemplo, para una rejila de 50×58 calcularemos 5692 incógnitas.

Usted organizará su aplicación en clases. El mapa deberá encapsularse en su propia clase, con métodos para cargar la imagen y construir la matriz $\mathbf{A}$ y el vector $\underline{\mathbf{b}}$ del sistema de ecuaciones.

```
namespace anpi {
  /// Pack a pair of indices of the nodes of a resistor
  struct indexPair {
    /// Row of the first node
    std::size_t row1,
```

```
    /// Column of the first node
    std::size_t col1,
    /// Row of the second node
    std::size_t row2
    /// Column of the second node
    std::size_t col2
  };


  class ResistorGrid {
  private:
    /// Matrix of the current equation system
    Matrix<float> A_;
    /// Vector of the current equation system
    std::vector<float> b_;

    /// Raw map data
    Matrix<float> rawMap_;
  public:
    /// ... constructors and other methods

    /**
     * Construct the grid from the given file
     * @return true if successful or false otherwise
     */
    bool build(const std::string filename);

    /**
     * Compute the internal data to navigate between the given nodes
     */
    bool navigate(const indexPair& nodes);
  };
}
```

1. El primer paso requerido es escribir un método de la clase `ResistorGrid` que mapee los índices de los dos nodos terminales de una resistencia a un índice lineal para acceder al vector $\underline{\mathbf{x}}$.

   ```
   namespace anpi {
     std::size_t ResistorGrid::nodesToIndex(const std::size_t row1,
                                            const std::size_t col1,
                                            const std::size_t row2,
                                            const std::size_t col2);
   }
   ```

   Si los índices de los nodos no son adjacentes, esta función debe emitir una excepción.

2. Se debe implementar además un método inverso al anterior que mapee un índice lineal de un elemento en $\underline{\mathbf{x}}$ a los índices de los nodos terminales de la resistencia.

   ```
   namespace anpi {
     /// Convert an index to the pair of node coordinates
     indexPair ResistorGrid::indexToNodes(const std::size_t idx);
   }
   ```

3. Escriba pruebas unitarias que verifiquen la validez de sus dos funciones que mapean índices.

4. Las primeras $n \times m$ ecuaciones que puede plantear son las ecuaciones de los nodos, en donde cada fila de $\mathbf{A}$ contendrá dos “1” y dos “-1” representando la suma de

las cuatro corrientes ilustradas en la figura 4, es decir, se suman las corrientes que entran por arriba y por la izquierda, y se restan las que salen abajo y a la derecha.

Note que para esta tarea usted requiere las funciones implementadas en los primeros puntos y además debe tener cuidado con los nodos en los bordes y esquinas, pues estos carecen de ciertos vecinos.

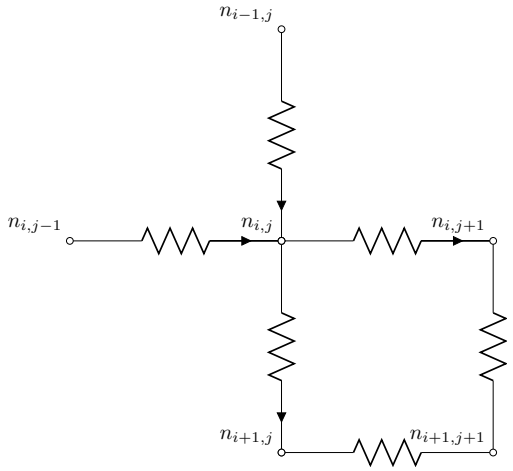

Figura 4: Nodo y malla de la rejilla para definir nomenclatura

El vector $\underline{\mathbf{b}}$ deberá ser inicializado de un tamaño igual al número de incógnitas y con valores cero, pues en todos los nodos la suma de corrientes es cero, excepto en dos de ellos: los correspondientes al nodo de inicio y nodo final. El componente de $\underline{\mathbf{b}}$ correspondiente al nodo de inicio será igual a la corriente de 1 A, y el componente correspondiente al nodo final deberá ser igual a -1 A.

Usted deberá descartar una de estas ecuaciones. La razón se explica más abajo, con el cuidado de verificar que la ecuación eliminada no sea ninguna correspondiente a los nodos de inicio o fin.

5. Para plantear el siguiente grupo de ecuaciones usted requiere el mapa especificado por el usuario en el método `build`. En el código entregado ya se enlaza la biblioteca OpenCV y en el archivo `src/paths.cpp` se encuentra un código ejemplo de cómo leer un archivo de imágen y cómo convertirlo a una `anpi::Matrix`.

   Asegúrese de probar su ejemplo con imágenes rectangulares (no cuadradas) para asegurar que no confunde filas y columnas (¡uno de los errores más usuales en el trabajo con matrices e imágenes!)

6. El siguiente grupo de ecuaciones a incorporar en la matriz $\mathbf{A}$ corresponde a las ecuaciones de malla, en los que deberá considerar que la suma de tensiones eléctricas en cada cuadrado de resistencias debe ser cero. Puesto que hay $(m-1) \times (n-1)$ cuadrados, se tienen $mn - (m+n) + 1$ ecuaciones adicionales.

   Observe que usted deberá colocar en la fila correspondiente a una ecuación de malla los cuatro valores de resistencia (con un signo positivo o negativo). Para elegir la

columna en la que corresponde una particular resistencia, debe entonces utilizar sus funciones de mapeo de índices. Recuerde que los valores de resistencia son dos posibles: uno para espacio libre y otro para espacio con obstáculos.

Junto con las $mn - (m + n)$ ecuaciones en el punto anterior se tienen con estas $2mn - (m + n) + 1$ ecuaciones: una ecuación de más para el número de incógnitas buscado.

Se recomienda eliminar la ecuación de nodo de una de las esquinas, con el cuidado de no eliminar ninguno de los nodos de inicio o fin, en caso de que el usuario haya colocado como inicio o fin a las esquinas. Si remueve una ecuación de malla, el sistema de ecuaciones no será consistente.

7. Ahora utilice su sistema de solución de ecuaciones para encontrar las corrientes en todas las resistencias.

   Note que la matrix $\mathbf{A}$ para un mapa de 50×58 nodos tendrá un tamaño de 5692×5692, ¡pues hay 5692 incógnitas! Cada matriz $\mathbf{A}$, $\mathbf{L}$ y $\mathbf{U}$ requerirá 124 MB de memoria RAM, sin considerar el relleno (*padding*).

   De nuevo, se recomienda vehementemente comprobar este código con mapas que no sean cuadrados, pues un error frecuente es intercambiar filas y columnas en las construcciones o en los accesos a las matrices.

8. Encuentre la primera trayectoria símplemente siguiendo las corrientes de salida máximas en cada nodo para pasar al siguiente, y repitiendo el proceso hasta llegar al destino.

   Es físicamente improbable que de un nodo salgan dos corrientes idénticas. Si ese caso ocurre, debe idear una estrategia para no caer en un bucle infinito.

9. Implemente alguna estrategia de visualización de esta trayectoria (ver por ejemplo la figura 5). Puede para ello extender la funcionalidad de la clase en `plotpy.h` para que permita visualizar los nodos y la trayectoria elegida, o puede implementar su propia clase.

   Como tarea opcional, puede sobreponer la trayectoria a la imagen del mapa, para poder visualizar mejor dónde están realmente los obstáculos.

10. Para la segunda estrategia de estimación de trayectoria, usted seguirá los siguientes pasos

    10.1. Calcule para cada nodo componentes (x, y) de desplazamiento, definidos símplemente como la suma de las dos corrientes horizontales x e y como la suma de las dos corrientes verticales, en los sentidos indicados en la figura 4.

    El sentido de este cálculo es crear vectores de desplazamiento en cada nodo. De este modo, si una corriente entra por la izquierda y sale por la derecha del nodo, quiere decir que en x hay un "movimiento" hacia la derecha. De modo similar, si una corriente entra por arriba y sale por abajo, entonces en y hay un desplazamiento hacia abajo. Las dos componentes dan una dirección neta

de movimiento, tal y como lo ilustra la figura 5 utilizando el método `quiver` de `matplotlib`.

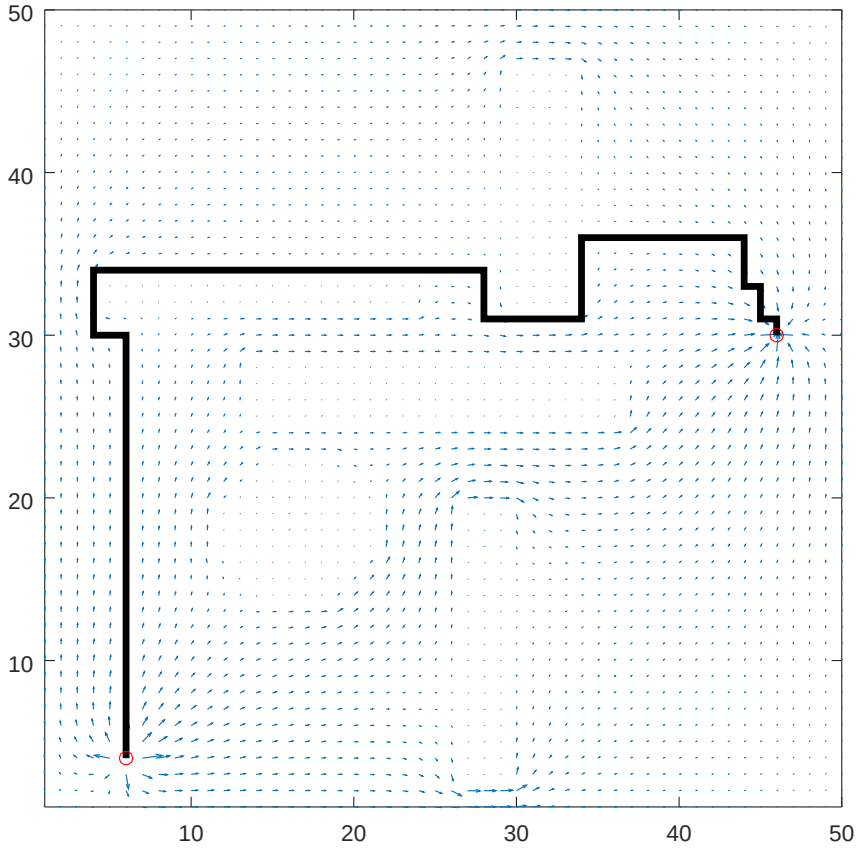

Figura 5: Campo de movimiento calculado en cada nodo. Se ilustra la trayectoria simple encontrada siguiendo las máximas corrientes.

Note que usted creará entonces dos matrices, una con los componentes x de los vectores de desplazamiento, y otra con los componentes y.

Usted deberá normalizar ambas matrices de modo que la magnitud máxima del vector de desplazamiento sea igual a 1.

El concepto de partícula funciona como sigue: la posición inicial de la partícula es $\underline{\mathbf{p}}_0$. Entonces se determina el vector de desplazamiento $\underline{\mathbf{d}}_0$ en esa posición $\underline{\mathbf{p}}_0$ y se suma para encontrar la nueva posición:

$$\underline{\mathbf{p}}_1 = \underline{\mathbf{p}}_0 + \alpha \underline{\mathbf{d}}_0$$

donde α es un parámetro que controla la velocidad de paso.

Ahora, debe determinarse el siguiente desplazamiento $\underline{\mathbf{d}}_1$ a partir de $\underline{\mathbf{p}}_1$, pero como esa posición ya probablemente no coincidirá con los índices enteros de un nodo de la rejilla resistiva, entonces debe utilizarse interpolación bilineal para encontrar los valores (x, y) del desplazamiento.

Este proceso se repite calculando

$$\underline{\mathbf{p}}_{i+1} = \underline{\mathbf{p}}_i + \alpha \underline{\mathbf{d}}_i$$

hasta llegar al destino con una determinada precisión.

10.2. Grafique sus trayectorias para tres valores distintos de α entre 0,1 y 1.

11. Realice un artículo formal, donde presente los resultados obtenidos. El esquema de un artículo formal lo puede encontrar aquí, en donde el énfasis debe darlo a las secciones de la propuesta para describir la estrategia de uso de SIMD y los resultados con las evaluaciones correspondientes. Dicho artículo no debe tener más de 3 páginas.

4 Entregables

El código fuente y el artículo científico deben ser entregados según lo estipulado en el programa del curso.

Incluya un archivo de texto README con las instrucciones para compilación y ejecución del programa.