

ICNPG 2023

Clase 9:  
Álgebra lineal Densa, Rala y Grafos...

# Algebra Lineal Numérica

- **CUFFT**: Una transformación lineal particular.
- **CUBLAS**: operaciones tipo I, II y III, optimizado para formatos de matriz densa.
- **NVBLAS**: si tu viejo programa C, octave, etc, usa BLAS, lo acelerás recompilando.
- **CUSPARSE**: operaciones de CUBLAS optimizadas para formatos ralos, etc.
- **CUSOLVER**: Resolución de Sistemas lineales, Autovalores, factorizaciones, etc.
- **NVGRAPH/CUGRAPH**: formatos y algoritmos para redes o grafos.
- **CUPY**: Python interfaces to many of the functions in the CUDA device/runtime, CUBLAS, CUFFT, and CUSOLVER.
- **MAGMA**: Como CUSOLVER+CUBLAS+CUSPARSE pero pensada para sistemas híbridos CPU/GPU.
- **Array-Fire**: librería de C++ library paralela con una API sencilla, para cpu, cuda y opencl!  
(y Python)

# A Common Library Workflow

Many CUDA libraries share concepts, features, and a common workflow when being called from a host application. A common workflow while using NVIDIA libraries is as follows:

1. Create a library-specific handle that manages contextual information useful for the library's operation.
2. Allocate device memory for inputs and outputs to the library function.
3. If inputs are not already in a library-supported format, convert them to be accessible by the library.
4. Populate the pre-allocated device memory with inputs in a supported format.
5. Configure the library computation to be executed.
6. Execute a library call that offloads the desired computation to the GPU.
7. Retrieve the results of that computation from device memory, possibly in a library-determined format.
8. If necessary, convert the retrieved data to the application's native format.
9. Release CUDA resources.  
*Si la librería es de alto nivel, esto se hace solo*
10. Continue with the remainder of the application.

# CuBLAS: Basic Linear Algebra Subprograms

- **Operaciones (números reales o complejos):**
  - Level I: functions that perform scalar and vector based operations.
  - Level II: functions that perform matrix-vector operations.
  - Level III: functions that perform matrix-matrix operations.
- **Cublas API:** Mas control, pero hay que manejar memoria del host y del device.
- **CublasXt API:** Menos control, pero solo hace falta manejar memoria del host.
- **Paralelismo:** soporta streams y multi-gpu, útiles para “batchear”. El precio a pagar es la comunicación entre las GPUs
- **Compilación:**
  - nvcc myCublasApp.c -lcublas -o myCublasApp  
para linkear con la librería
  - g++ myCublasApp.c -lcublas\_static -lcublas -lcudart\_static -lphthread -ldl -I <cuda-toolkit-path>/include -L <cuda-toolkit-path>/lib64 -o myCublasApp
- **Ejemplo (level III):**  
[/share/apps/icnpg/clases/Cuda\\_Basico/MultMat/soluciones/multmat\\_solucion.cu](/share/apps/icnpg/clases/Cuda_Basico/MultMat/soluciones/multmat_solucion.cu)

*Multiplicacion de Matrices de 1024x1024 en nuestro cluster*

naive CPU:	6023.591623 ms	(1x)
naive GPU:	120.343554 ms	(50x)
CUBLASXT:	17.587757 ms	(300x)
CUBLAS:	9.224054 ms	(600x)

# CuBLAS levels

scalar and vector based operations

## ▽ 2.5. cuBLAS Level-1 Function Reference

2.5.1. cublasI<T>amax()
2.5.2. cublasI<T>amin()
2.5.3. cublas<T>asum()
2.5.4. cublas<T>axpy()
2.5.5. cublas<T>copy()
2.5.6. cublas<T>dot()
2.5.7. cublas<T>nrm2()
2.5.8. cublas<T>rot()
2.5.9. cublas<T>rotg()
2.5.10. cublas<T>rotm()
2.5.11. cublas<T>rotmg()
2.5.12. cublas<T>scal()
2.5.13. cublas<T>swap()

matrix-vector operations.

(mv)

## ▽ 2.6. cuBLAS Level-2 Function Reference

2.6.1. cublas<T>gbmv()
2.6.2. cublas<T>gemv()
2.6.3. cublas<T>ger()
2.6.4. cublas<T>sbmv()
2.6.5. cublas<T>spmv()
2.6.6. cublas<T>spr()
2.6.7. cublas<T>spr2()
2.6.8. cublas<T>symv()
2.6.9. cublas<T>syr()
2.6.10. cublas<T>syr2()
2.6.11. cublas<T>tbbmv()
2.6.12. cublas<T>tbsv()
2.6.13. cublas<T>tpmv()
2.6.14. cublas<T>tpsv()
2.6.15. cublas<T>trmv()
2.6.16. cublas<T>trsv()
2.6.17. cublas<T>hemv()
2.6.18. cublas<T>hbbmv()
2.6.19. cublas<T>hpmv()
2.6.20. cublas<T>her()
2.6.21. cublas<T>her2()
2.6.22. cublas<T>hpr()
2.6.23. cublas<T>hpr2()

matrix-matrix operations.

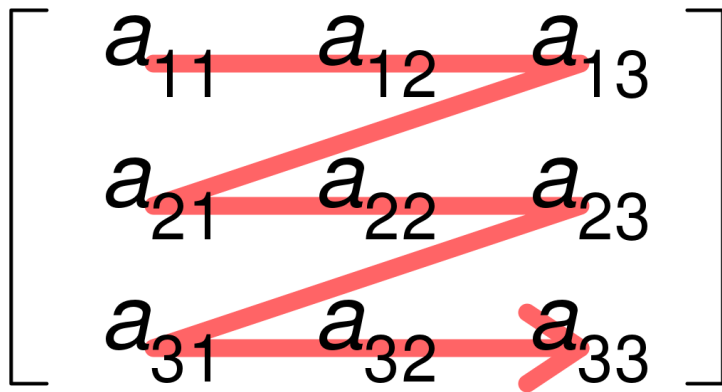
## ▽ 2.7. cuBLAS Level-3 Function Reference

2.7.1. cublas<T>gemm()
2.7.2. cublas<T>gemm3m()
2.7.3. cublas<T>gemmBatched()
2.7.4. cublas<T>gemmStridedBatched()
2.7.5. cublas<T>symm()
2.7.6. cublas<T>syrk()
2.7.7. cublas<T>syr2k()
2.7.8. cublas<T>syrkx()
2.7.9. cublas<T>trmm()
2.7.10. cublas<T>trsm()
2.7.11. cublas<T>trsmBatched()
2.7.12. cublas<T>hemm()
2.7.13. cublas<T>herk()
2.7.14. cublas<T>her2k()
2.7.15. cublas<T>herkx()

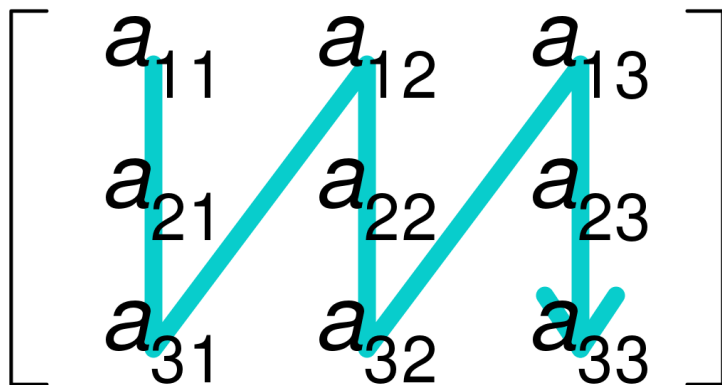
# Data layout

Todas las librerías aceptan matrices ordenadas linealmente, solo hay que decirle si están ordenadas column major o row major. CuBLAS asume column major

## Row-major order



## Column-major order



### 1.1. Data layout

For maximum compatibility with existing Fortran environments, the cuBLAS library uses column-major storage, and 1-based indexing. Since C and C++ use row-major storage, applications written in these languages can not use the native array semantics for two-dimensional arrays. Instead, macros or inline functions should be defined to implement matrices on top of one-dimensional arrays. For Fortran code ported to C in mechanical fashion, one may choose to retain 1-based indexing to avoid the need to transform loops. In this case, the array index of a matrix element in row “i” and column “j” can be computed via the following macro

```
#define IDX2F(i,j,ld) (((j)-1)*(ld))+((i)-1))
```

Here, ld refers to the leading dimension of the matrix, which in the case of column-major storage is the number of rows of the allocated matrix (even if only a submatrix of it is being used).

For natively written C and C++ code, one would most likely choose 0-based indexing, in which case the array index of a matrix element in row “i” and column “j” can be computed via the following macro

```
#define IDX2C(i,j,ld) ((j)*(ld))+i)
```

# CuSPARSE: cuBLAS para matrices ralas

*The cuSPARSE library contains a set of GPU-accelerated basic linear algebra subroutines used for handling sparse matrices that perform significantly faster than CPU-only alternatives. Depending on the specific operation, the library targets matrices with sparsity ratios in the range between 70%-99.9%.*

sparse matrix = matriz ralas

- > Operations between a **sparse vector** and a **dense vector**: sum, dot product, scatter, gather
- > Operations between a **dense matrix** and a **sparse vector**: multiplication
- > Operations between a **sparse matrix** and a **dense vector**: multiplication, triangular solver, tridiagonal solver, pentadiagonal solver
- > Operations between a **sparse matrix** and a **dense matrix**: multiplication, triangular solver, tridiagonal solver, pentadiagonal solver
- > Operations between a **sparse matrix** and a **sparse matrix**: sum, multiplication
- > Operations between **dense matrices** with output a **sparse matrix**: multiplication
- > **Sparse matrix preconditioners**: Incomplete Cholesky Factorization (level 0), Incomplete LU Factorization (level 0)
- > Reordering and Conversion operations between different **sparse matrix storage formats**
- **Formateo**: operations that allow conversion between different matrix formats, and compression of **csr** matrices.  
↳ cimpresed-sparsed-rows
- **Compilación**:
  - `nvcc myCusparsingApp.c -lcusparsing -o myCusparsingApp`
  - `g++ myCusparsingApp.c -lcusparsing_static -lcublas -lcudart_static -lpthread -ldl -I <cuda-toolkit-path>/include -L <cuda-toolkit-path>/lib64 -o myCusparsingApp`
- **Ejemplos**: copiar del toolkit...

# CuSPARSE: Formatos de vectores

- Vectores Densos

Se guarda todo, inclusive los valores nulos

1.0 0.0 0.0 2.0 3.0 0.0 4.0

single data array that is stored linearly in memory

- Vectores Ralos

Se guardan los elementos distintos de cero y las posiciones

1.0 2.0 3.0 4.0  
0 3 4 6

The data array has the nonzero values from the equivalent array in dense format.

The integer index array has the positions of the corresponding nonzero values in the equivalent array in dense format.



# CuSPARSE: Formatos de Matrices

- Matrices Densas

$$\begin{matrix} X_{1,1} & X_{1,2} & \cdots & X_{1,n} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m,1} & X_{m,2} & \cdots & X_{m,n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{ldX,1} & X_{ldX,2} & \cdots & X_{ldX,n} \end{matrix}$$

The dense matrix  $X$  is assumed to be stored in **column-major format in memory**

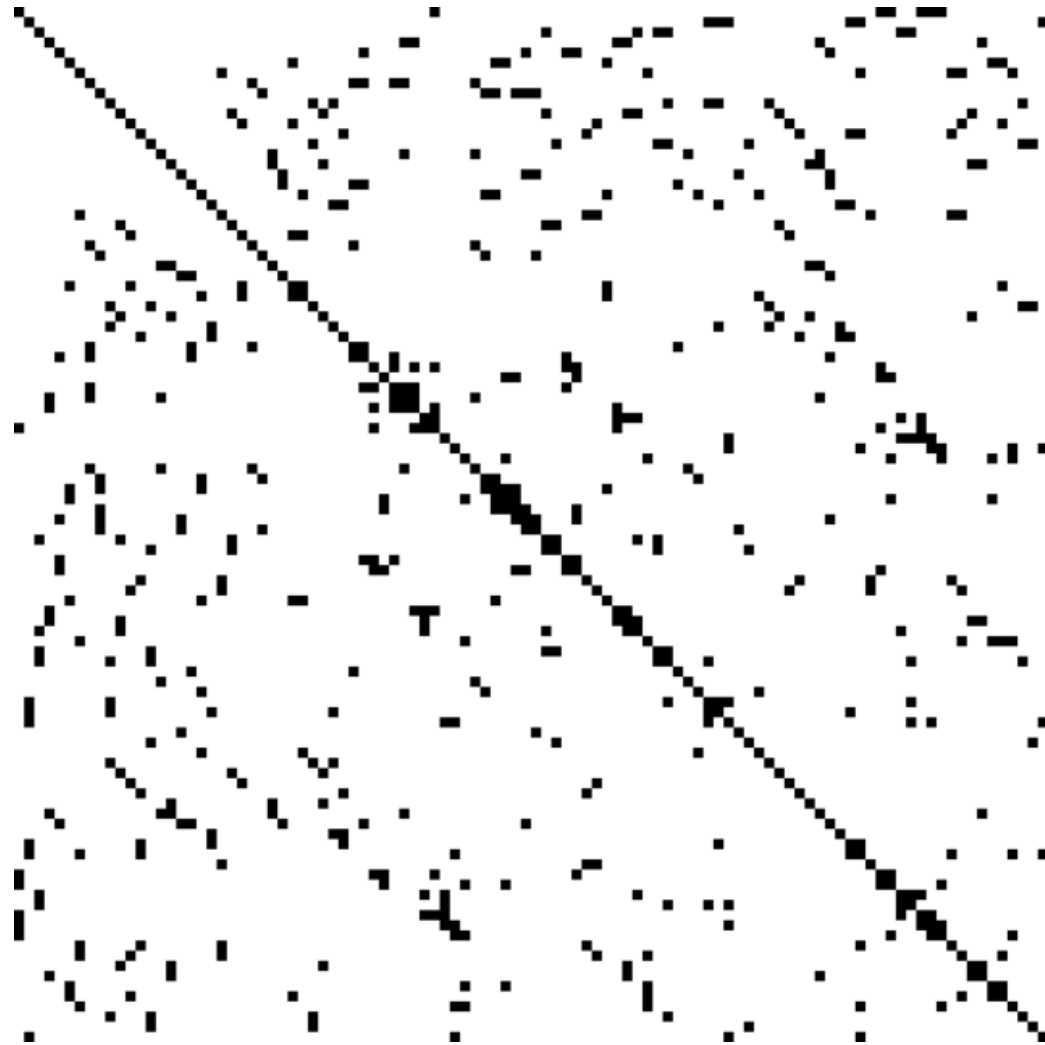
$$X_{1,1} \quad X_{2,1} \quad \cdots \quad X_{m,1} \quad \cdots \quad X_{ldX,1} \quad \cdots \quad X_{1,n} \quad X_{2,n} \quad \cdots \quad X_{m,n} \quad \cdots \quad X_{ldX,n}$$

and is represented by the following parameters.

<code>m</code>	(integer)	The number of rows in the matrix.
<code>n</code>	(integer)	The number of columns in the matrix.
<code>ldX</code>	(integer)	The leading dimension of <code>x</code> , which must be greater than or equal to <code>m</code> . If <code>ldX</code> is greater than <code>m</code> , then <code>x</code> represents a sub-matrix of a larger matrix stored in memory, <i>es decir, uno podría trabajar con una submatriz</i>
<code>x</code>	(pointer)	Points to the data array containing the matrix elements. It is assumed that enough storage is allocated for <code>x</code> to hold all of the matrix elements and that cuSPARSE library functions may access values outside of the sub-matrix, but will never overwrite them.

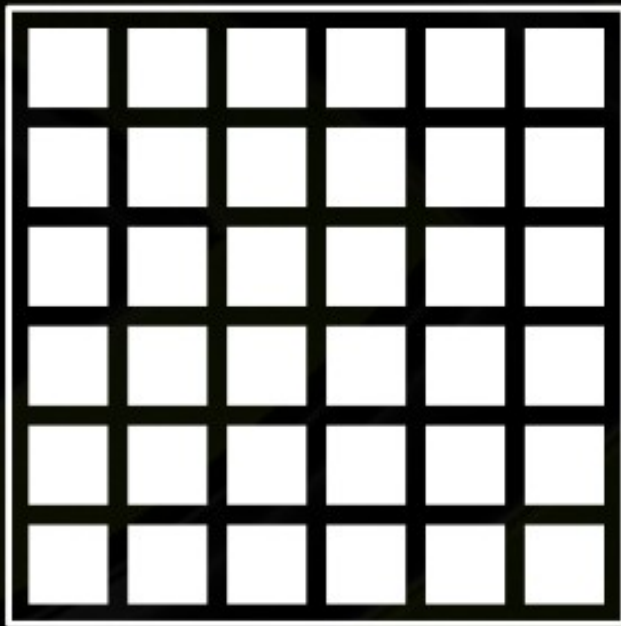
# CuSPARSE: Formatos de Matrices

- Matrices Ralas

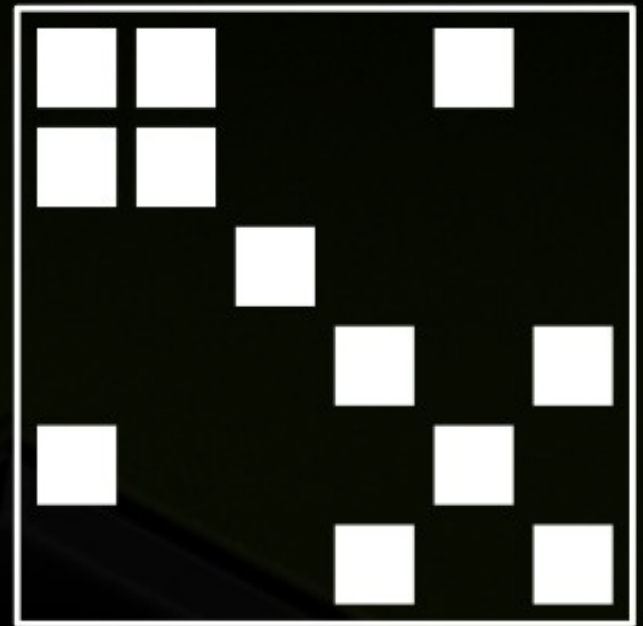


# Que son las matrices ralas (sparse matrix)?

- **Have small number of non-zero entries**



Dense Matrix



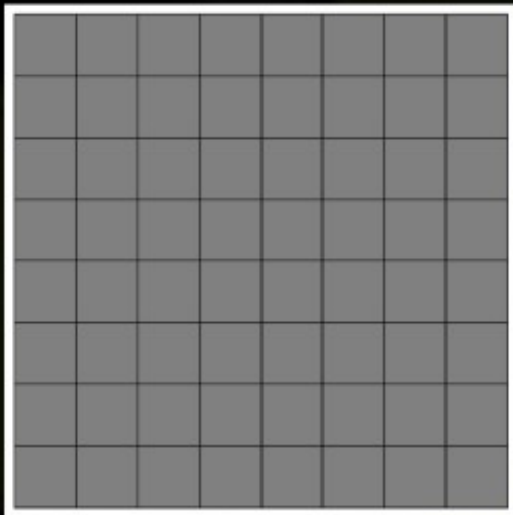
Sparse Matrix

# Donde aparecen las matrices ralas?

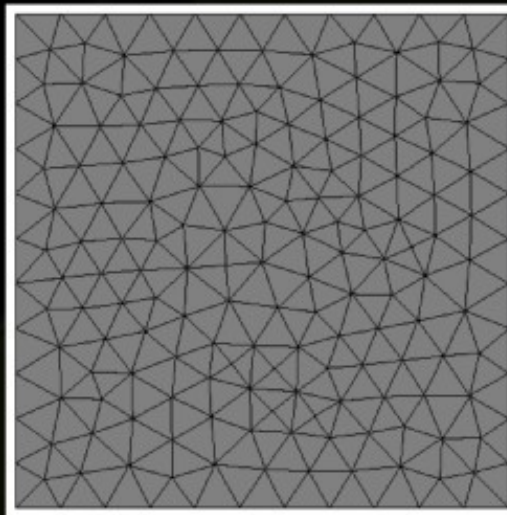
- **Non-zeros encode connectivity**

- **Finite-Element Meshes, Hyperlinks, Social Networks, ...**

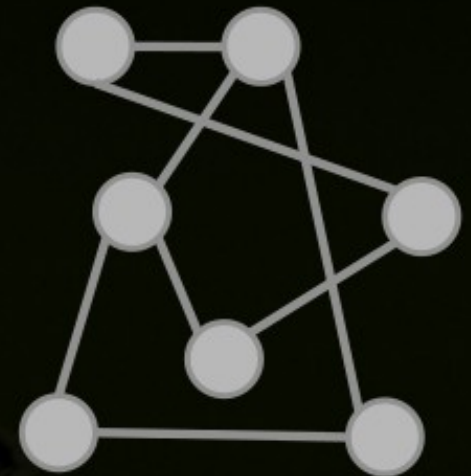
La matriz de conectividad es rala



Structured Mesh

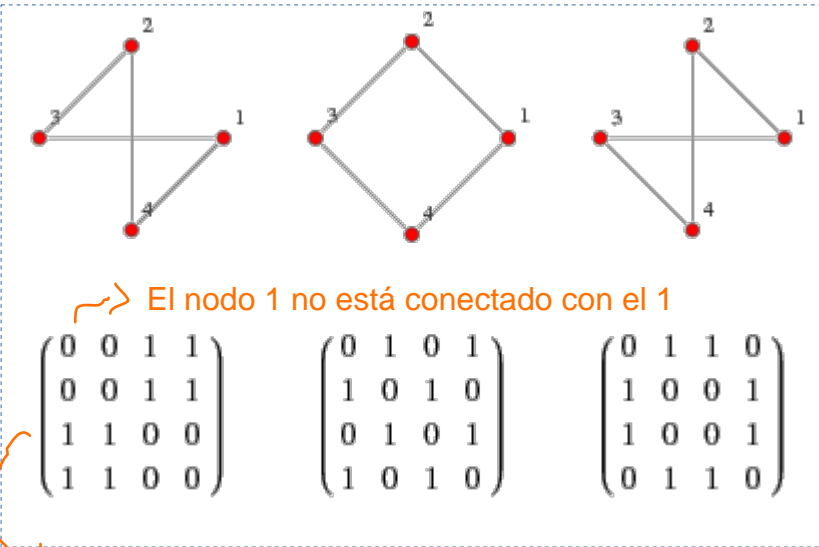


Unstructured Mesh

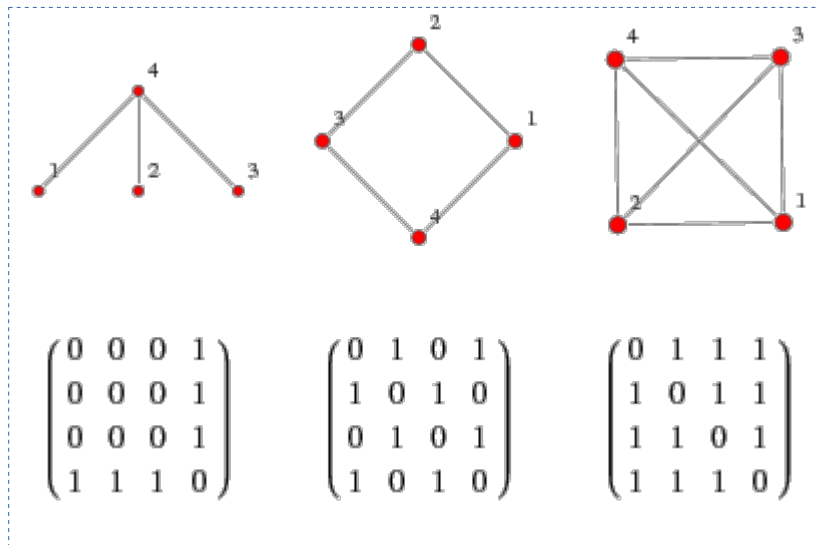


General Graph

# Matriz de conectividad o adyacencia



El nodo 3 está conectado con el 1

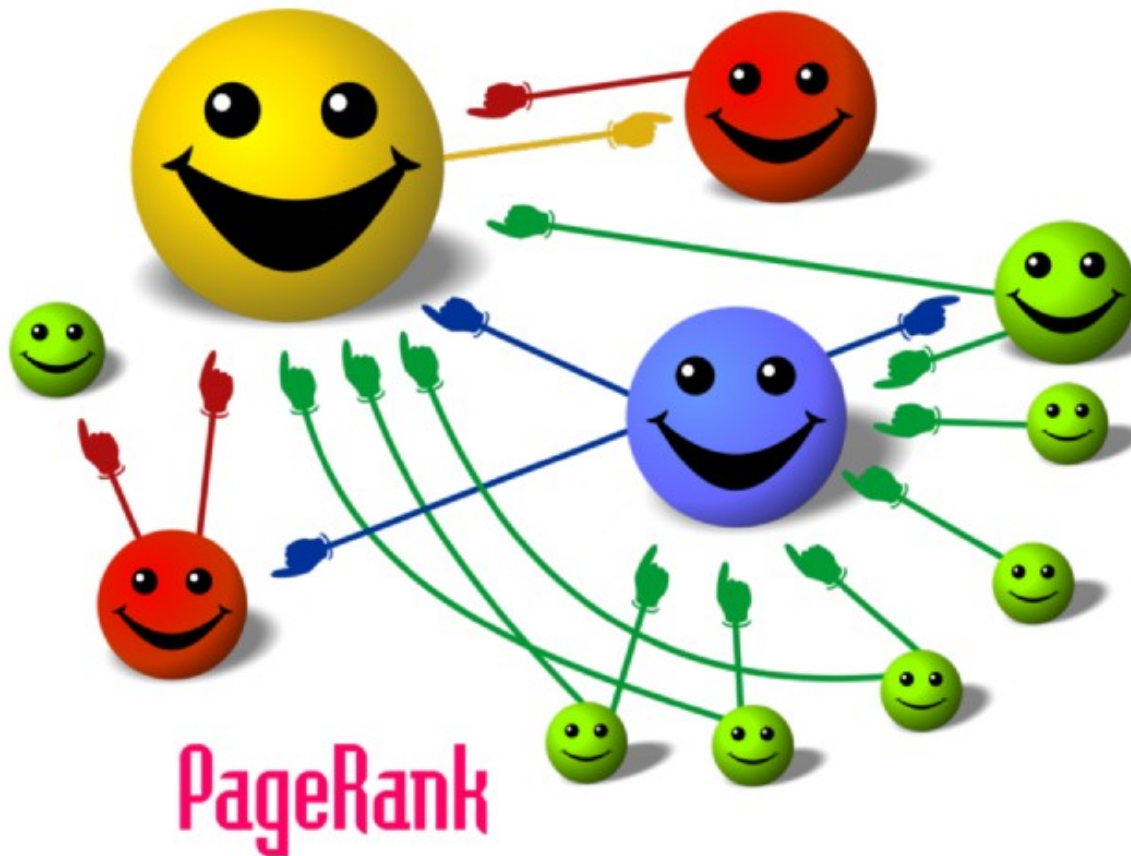


La matriz es basicamente una estructura de datos muy útil para representar y manipular grafos en programas de computadora.

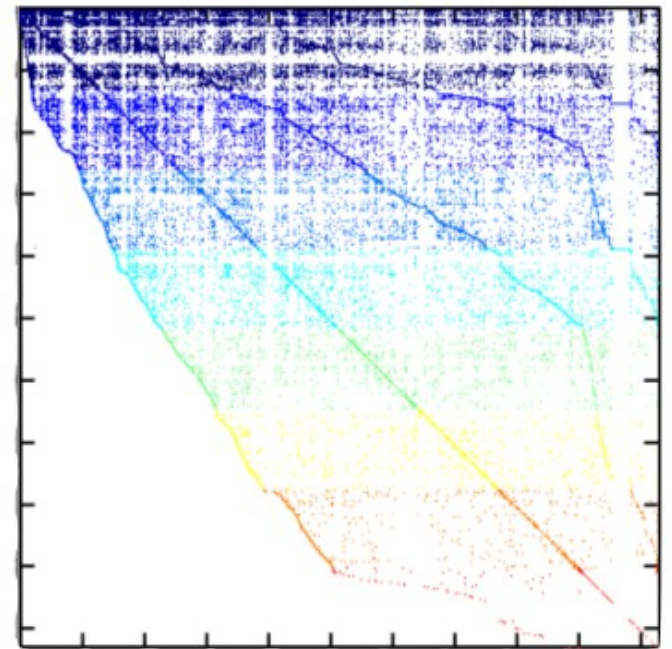
The adjacency matrix, sometimes also called the connection matrix, of a simple labeled graph is a **matrix** with rows and columns labeled by **graph vertices**, with a 1 or 0 in position according to whether and are **adjacent** or not. For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal. For an **undirected graph**, the adjacency matrix is **symmetric**.

# Matrices: Page Rank (Brin, Page)

Ranking de páginas que da lugar a un buscador web (como Google)



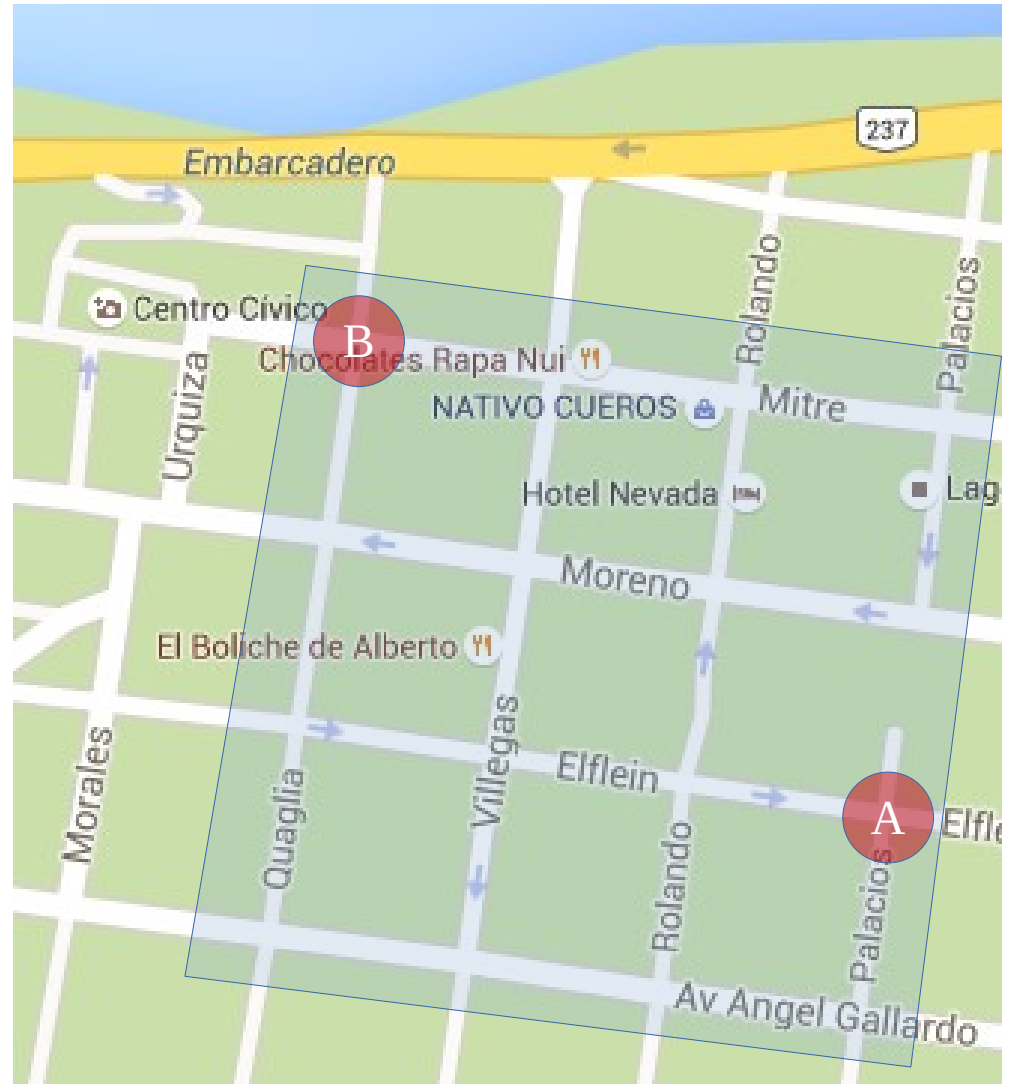
$$A_{ij}=1 \text{ si } i \rightarrow j$$



# Ejercicio

- Mitre, Moreno, Elflein, Gallardo.
- Quaglia, Villegas, Rolando, Palacios.
- *Salgo de (A) Elflein y Palacios*
- *Llego a (B) Mitre y Quaglia*
- *¿camino mas corto?*
- *¿cuantos caminos de  $n$  cuadras hay?*
- *¿Cuantos caminos de hasta  $n$  cuadras?*
- *¡A pie no es igual que en auto!*
- *Etc.*

Todo esto se puede responder usando teoría de grafos





# Matriz de conectividad en auto (2016!)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1																
2																
3																
4																
5																
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																
16																





# Matriz de conectividad en auto

No simétrica

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		■														
2			■			■										
3				■												
4							■									
5	■															
6					■					■						
7			■			■										
8							■									
9					■					■						
10										■				■		
11							■				■					
12																■
13								■						■		
14										■					■	
15											■			■		■
16															■	



# ¿Matriz de conectividad a pie?

Simétrica

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		■			■											
2	■		■			■										
3		■		■												
4			■					■								
5	■					■			■							
6		■			■		■			■						
7			■			■		■			■					
8				■			■									
9					■					■			■			
10						■			■		■			■		
11							■			■		■			■	
12										■						■
13									■				■	■		
14										■			■		■	
15											■		■	■		■
16												■			■	



# Test de vigilia ...

Simétrica

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		■			■											
2	■		■			■										
3		■		■												
4			■					■								
5	■					■			■							
6		■			■		■			■						
7			■			■		■			■					
8				■			■									
9					■					■			■			
10						■			■		■			■		
11							■			■		■			■	
12										■						■
13								■					■	■		
14										■		■		■	■	
15											■		■		■	■
16												■			■	

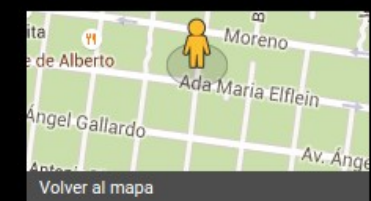


# Conectividad a pie...

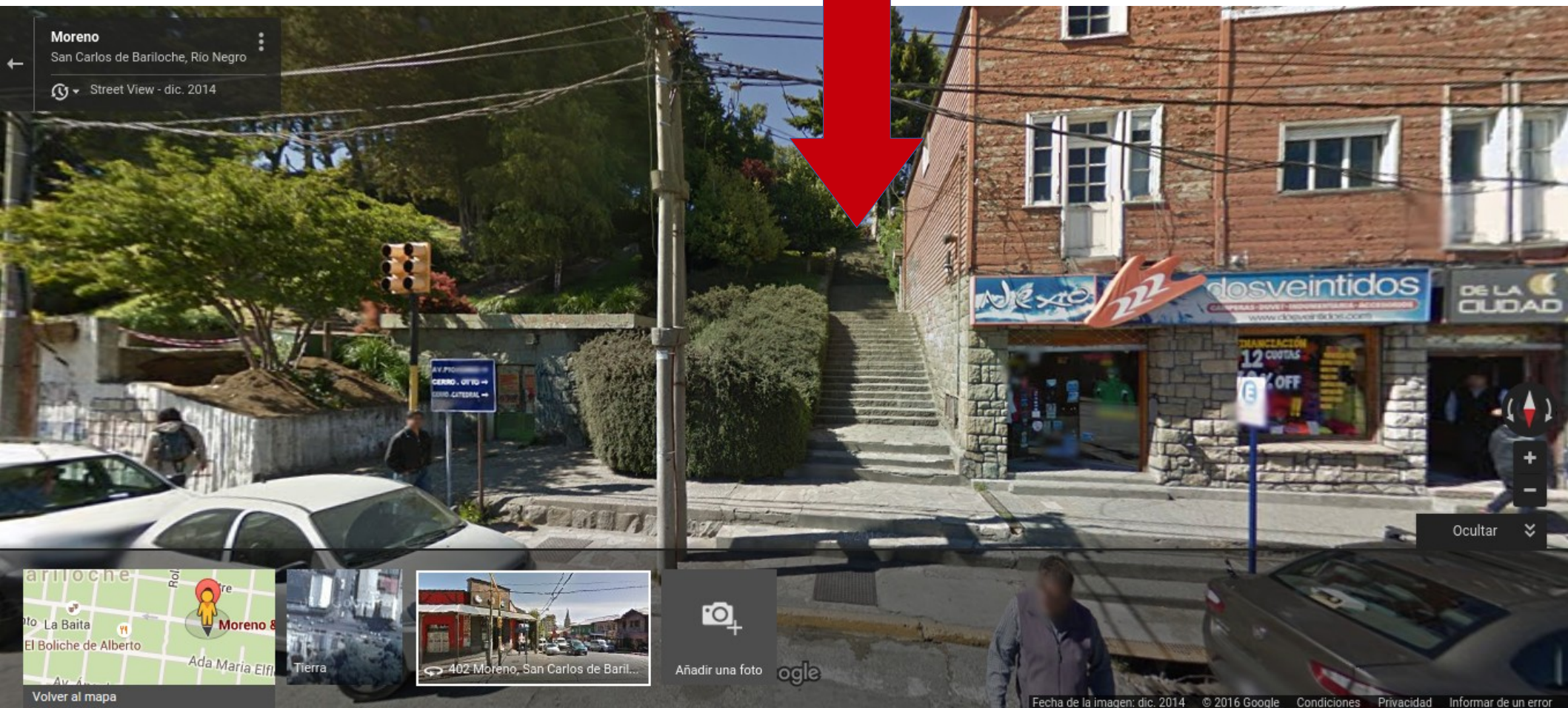








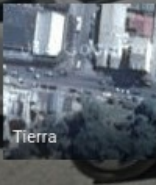
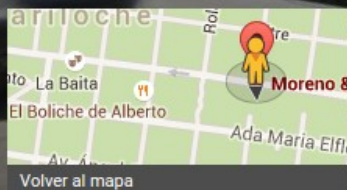




Moreno

San Carlos de Bariloche, Río Negro

Street View - dic. 2014



Añadir una foto

Google

Fecha de la imagen: dic. 2014 © 2016 Google Condiciones Privacidad Informar de un error

# ¿Matriz de conectividad a pie?

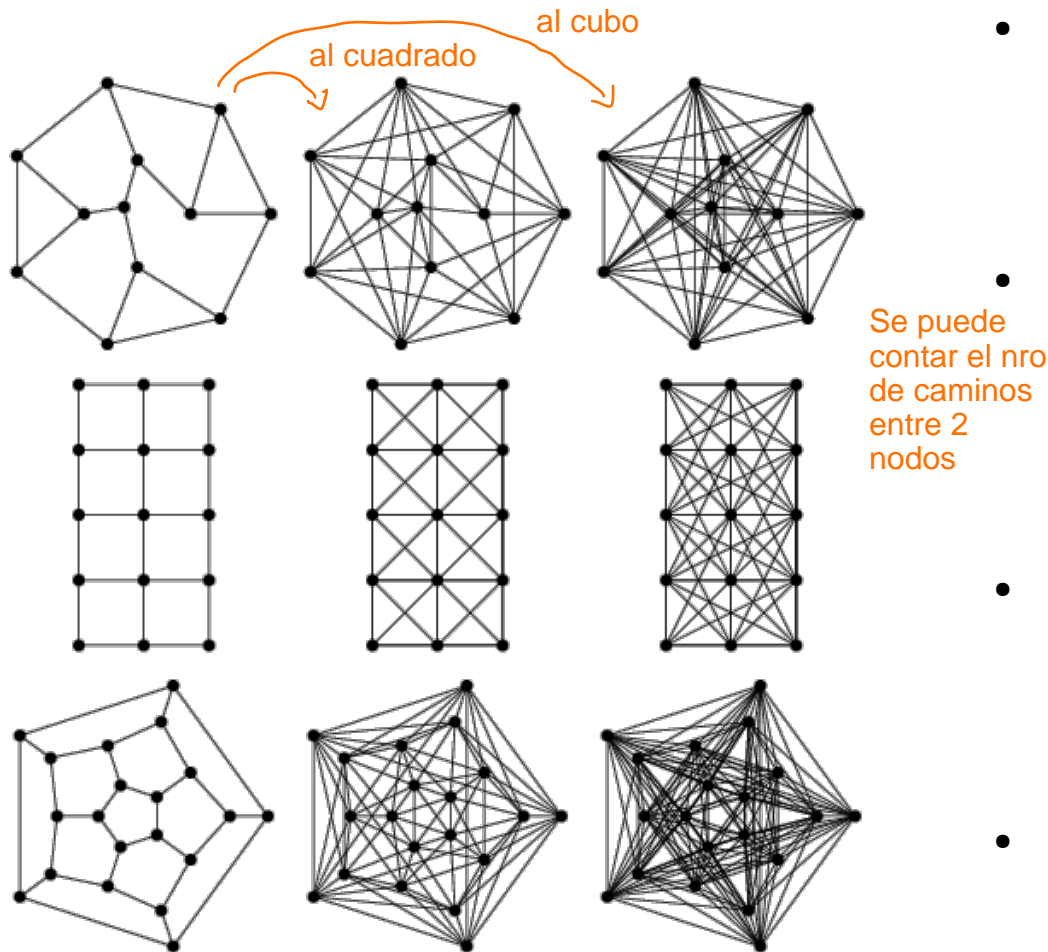
simétrica

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		■			■											
2	■		■			■										
3		■		■												
4			■					■								
5	■					■			■							
6		■			■		■			■						
7			■			■		■			■					
8				■			■					■				
9					■					■			■			
10						■			■		■			■		
11							■			■		■			■	
12								■			■					■
13									■				■			
14										■			■		■	
15											■			■		■
16												■			■	





# Propiedades de la matriz de adyacencia



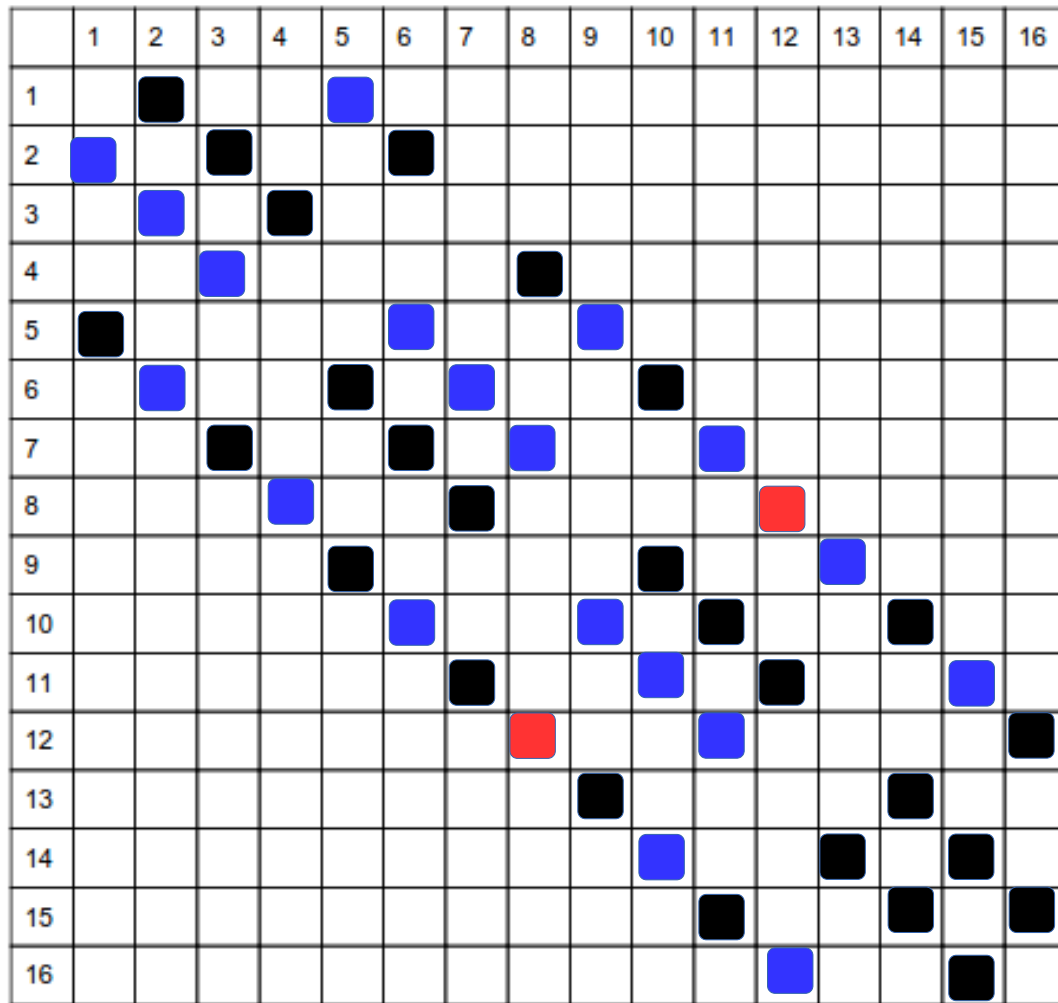
- The  $k$ th power of a graph  $G$  is a graph with the same set of vertices as  $G$  and an edge between two vertices iff there is a path of length at most  $k$  between them.
- Since a path of length two between vertices  $u$  and  $v$  exists for every vertex  $w$  such that  $\{u,w\}$  and  $\{w,v\}$  are edges in  $G$ , the square of the adjacency matrix of  $G$  counts the number of such paths.
- Similarly, the  $(u,v)$ th element of the  $k$ th power of the adjacency matrix of  $G$  gives the number of paths of length  $k$  between vertices  $u$  and  $v$ .
- The graph  $k$ th power is then defined as the graph whose adjacency matrix given by the sum of the first  $k$  powers of the adjacency matrix,

$$\text{adj}(G^k) = \sum_{i=1}^k [\text{adj}(G)]^i,$$

which counts all paths of length up to  $k$ .

Camino más corto, entre uno o más puntos, o número de caminos de  $k$  pasos entre dos puntos, etc, etc.

Simétrica y rala (sparse)



# Ecuación de Laplace

- Ecuación de difusión
- Ecuación del calor
- Membrana elástica
- Electrostática...
- Etc.

$$V = f_N(x)$$

$$(\partial_x^2 + \partial_y^2)V = 0$$

$$V = f_O(y)$$

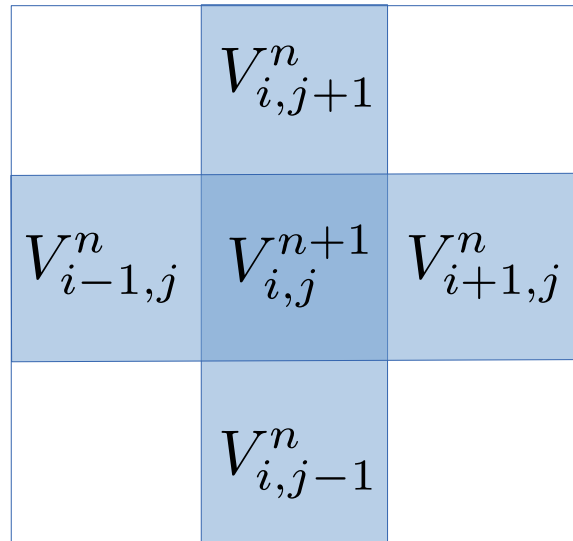
$$V(x, y) = ?$$

$$V = f_E(y)$$

$$V = f_S(x)$$

# Método de Jacobi

$$V_{i,j}^{n+1} = \frac{1}{4}(V_{i+1,j}^n + V_{i-1,j}^n + V_{i,j+1}^n + V_{i,j-1}^n)$$



# Método de Jacobi

$$V_{i,j}^{n+1} = \frac{1}{4}(V_{i+1,j}^n + V_{i-1,j}^n + V_{i,j+1}^n + V_{i,j-1}^n)$$

*En algebra lineal numérica toda matriz es un vector*

$$\alpha(i, j) = i + jL = 0, 1, \dots, L^2 - 1 \rightarrow V_{\alpha}^{n+1} = \sum_{\beta} A_{\alpha, \beta} V_{\beta}^n$$

# Método de Jacobi

$$V_{i,j}^{n+1} = \frac{1}{4}(V_{i+1,j}^n + V_{i-1,j}^n + V_{i,j+1}^n + V_{i,j-1}^n)$$

$$V_{\alpha}^{n+1} = \sum_{\beta} A_{\alpha,\beta} V_{\beta}^n$$

$$\mathbf{V}^{n+1} = \mathbf{A} \cdot \mathbf{V}^n$$

¿Cuántos elementos distintos de cero tiene A?

A matriz real de  $L^2 \times L^2$  rala

# Convolución o Correlación

$$y_i = \sum_k h_k x_{k+i} \rightarrow \mathbf{y} = \mathbf{H} \cdot \mathbf{x}$$

¿Cuántos elementos distintos de cero tiene  $\mathbf{H}$  si el filtro es local?

*El filtro puede representar un operador diferencial discretizado*

Por ejemplo en 1D

Esto se llama un "extensil"

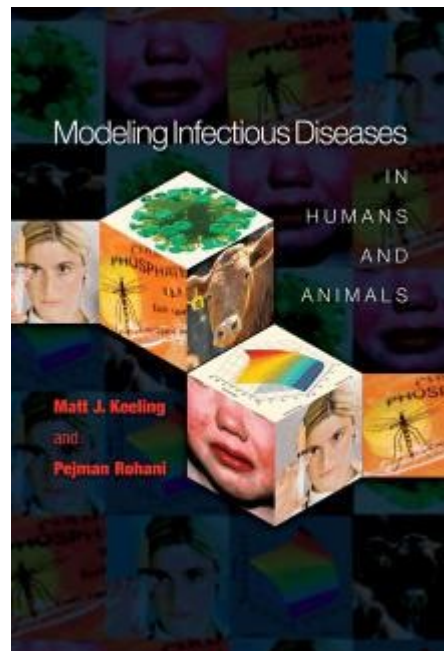


$$\mathbf{h} = \{1, -2, 1\} / \Delta x^2 \rightarrow$$

$$y_i = \sum_k h_k u_{k+i} = \frac{u_{i+1} + u_{i-1} - 2u_i}{\Delta x^2} \equiv \partial_x^2 u$$

En general los operadores diferenciales locales dan una  $\mathbf{H}$  rala.

# Ejemplo: epidemias



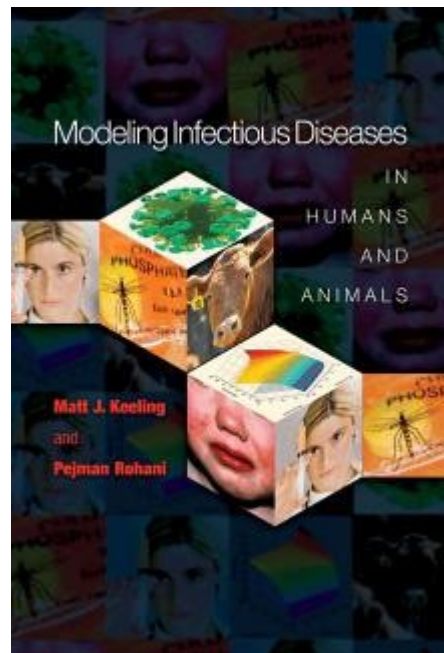
$$\frac{dS}{dt} = \mu - \beta SI - \mu S,$$

$$\frac{dI}{dt} = \beta SI - \gamma I - \mu I,$$

$$\frac{dR}{dt} = \gamma I - \mu R.$$



# Ejemplo: epidemias



$$\frac{dX_i}{dt} = v_i N_i - \lambda_i X_i - \mu_i X_i,$$

$$\frac{dY_i}{dt} = \lambda_i X_i - \gamma_i Y_i - \mu_i Y_i,$$

$$\lambda_i = \beta_i \sum_j \rho_{ij} \frac{X_j}{N_i},$$

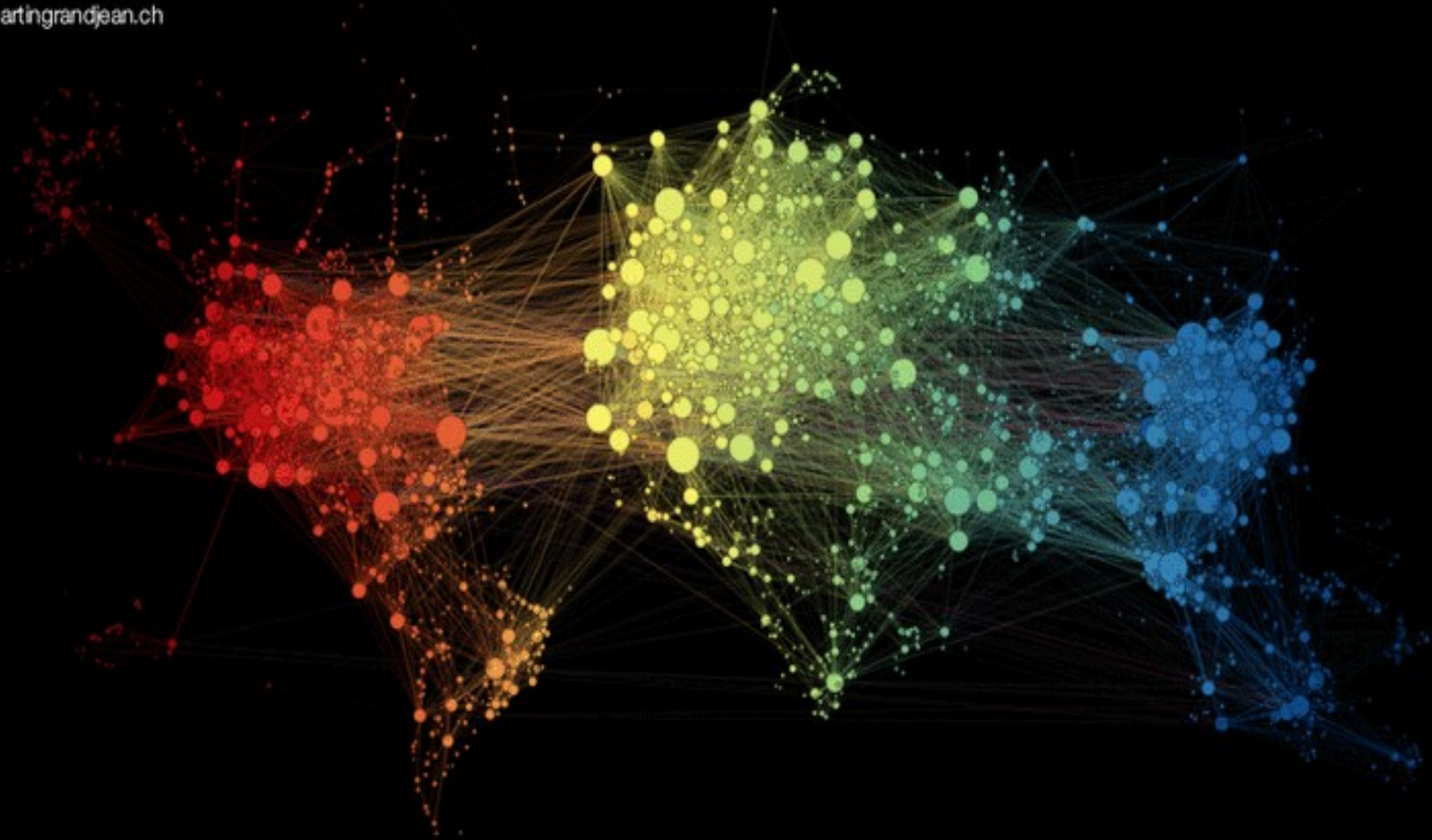


Elementos de matriz  
Típicamente rala

# CONNECTIONS

3.200 AIRPORTS

[martingrandjean.ch](http://martingrandjean.ch)



Matriz de adyacencia rara...

*Viajan personas... portando enfermedades, mercancías, etc!*

## ● Coordinate (COO)

1	7	0	0
0	2	8	0
5	0	3	9
0	6	0	4

Matrix

0 0 1 1 2 2 2 3 3

row indices

0 1 1 2 0 2 3 1 3

column indices

1 7 2 8 5 3 9 6 4

values

# entries

# Formatos para matrices ralas

- **Compressed Sparse Row (CSR)**

1	7	0	0
0	2	8	0
5	0	3	9
0	6	0	4

Matrix

# rows + 1

0	2	4	7	9
---	---	---	---	---

row offsets

0	1	1	2	0	2	3	1	3
---	---	---	---	---	---	---	---	---

column indices

1	7	2	8	5	3	9	6	4
---	---	---	---	---	---	---	---	---

values

# entries

# Formatos para matrices ralas

## ● ELLPACK (ELL)

1	7	0	0
0	2	8	0
5	0	3	9
0	6	0	4

Matrix

# entries per row

0	1	*
1	2	*
0	1	2
1	3	*

column indices

padding

1	7	*
2	8	*
5	3	9
6	4	*

values

# rows



# Formatos para matrices ralas

## ● Diagonal (DIA)

1	7	0	0
0	2	8	0
5	0	3	9
0	6	0	4

Matrix

# diagonals

*	1	7
*	2	8
5	3	9
6	4	*

# rows

values

# Formatos para matrices ralas

## ● Hybrid (HYB) ELL + COO

1	7	0	0
0	2	8	0
5	0	3	9
0	6	0	4

Matrix

ELL

0	1
1	2
0	1
1	3

column indices

1	7
2	8
5	3
6	4

values

COO

2

row indices

3

column indices

9

values

# Formatos para matrices ralas

Esta es la que más se usa

(DIA) Diagonal

(ELL) ELLPACK

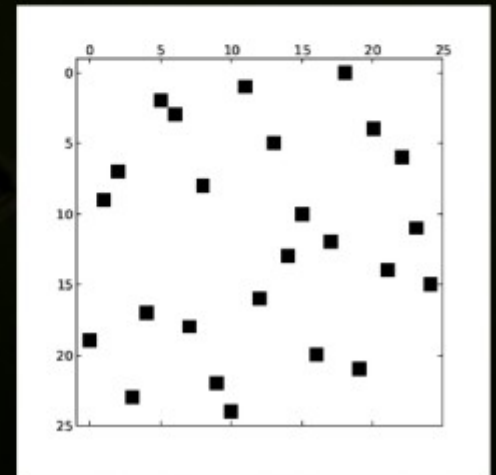
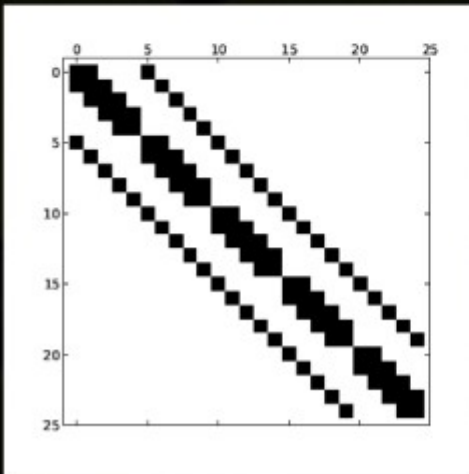
(CSR) Compressed Row

(HYB) Hybrid

(COO) Coordinate

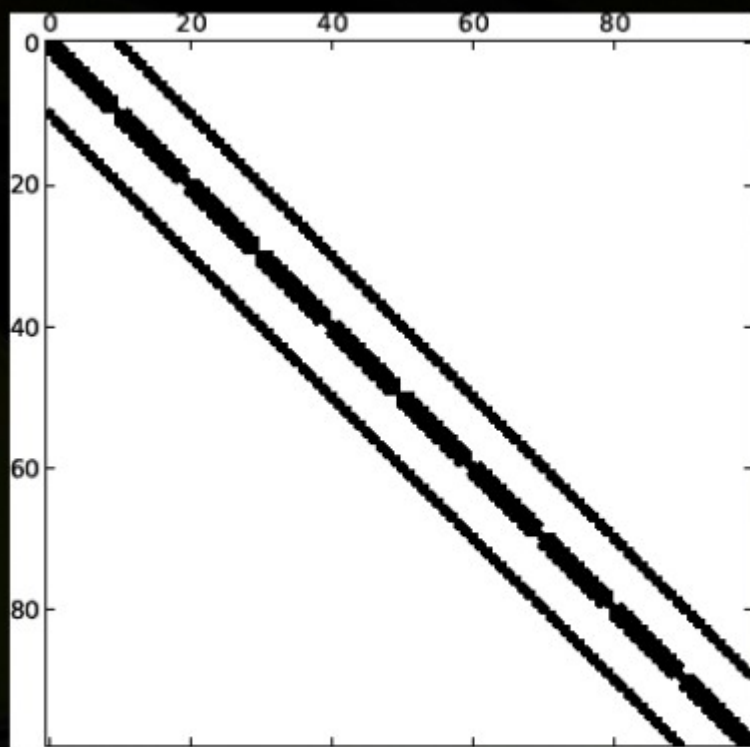
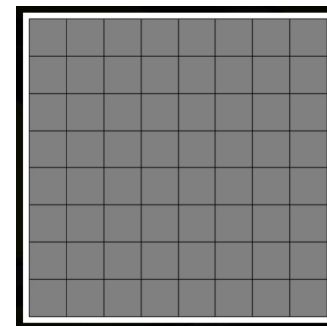
Structured

Unstructured





# Que formato elegir?

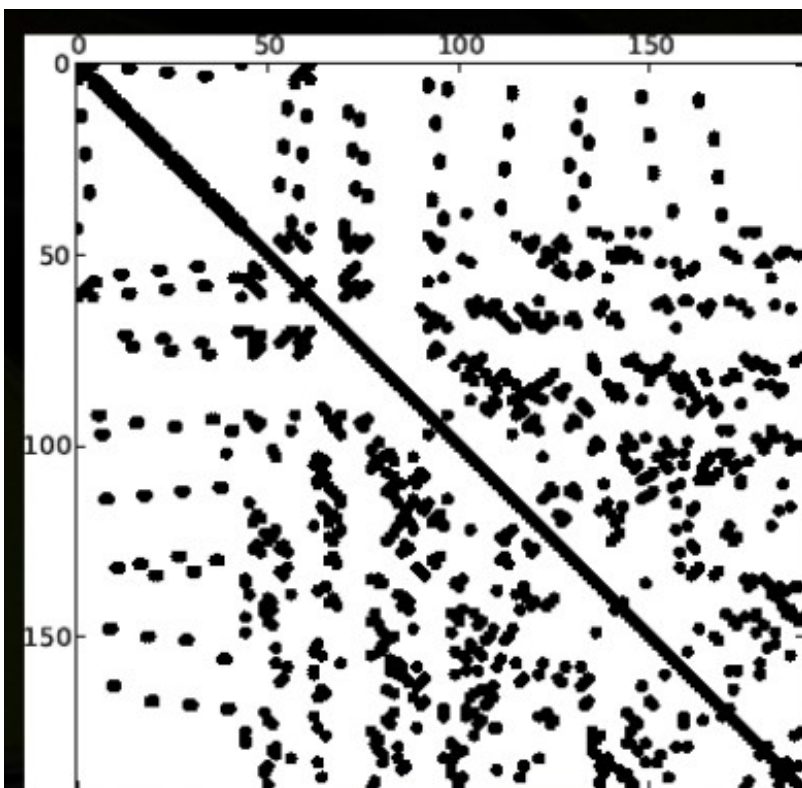
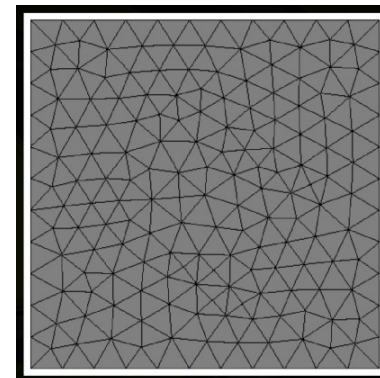


Matrix

Format	float	double
COO	12.00	16.00
CSR	8.45	12.45
DIA	4.05	8.10
ELL	8.11	12.16
HYB	8.11	12.16

Bytes per Nonzero Entry

# Que formato elegir?



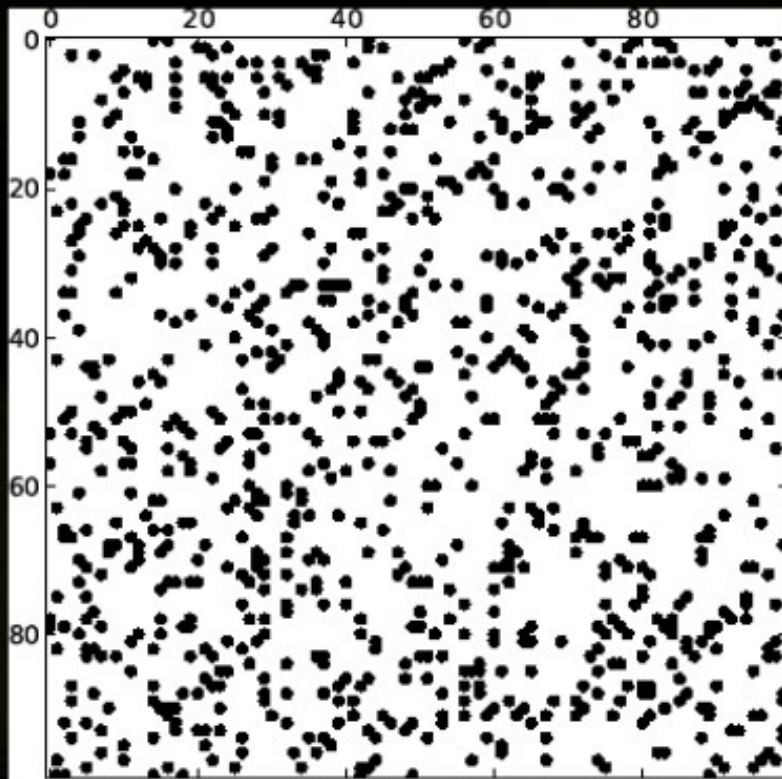
Matrix

Format	float	double
COO	12.00	16.00
CSR	8.62	12.62
DIA	164.11	328.22
ELL	11.06	16.60
HYB	9.00	13.44

Bytes per Nonzero Entry

# Que formato elegir?

Random



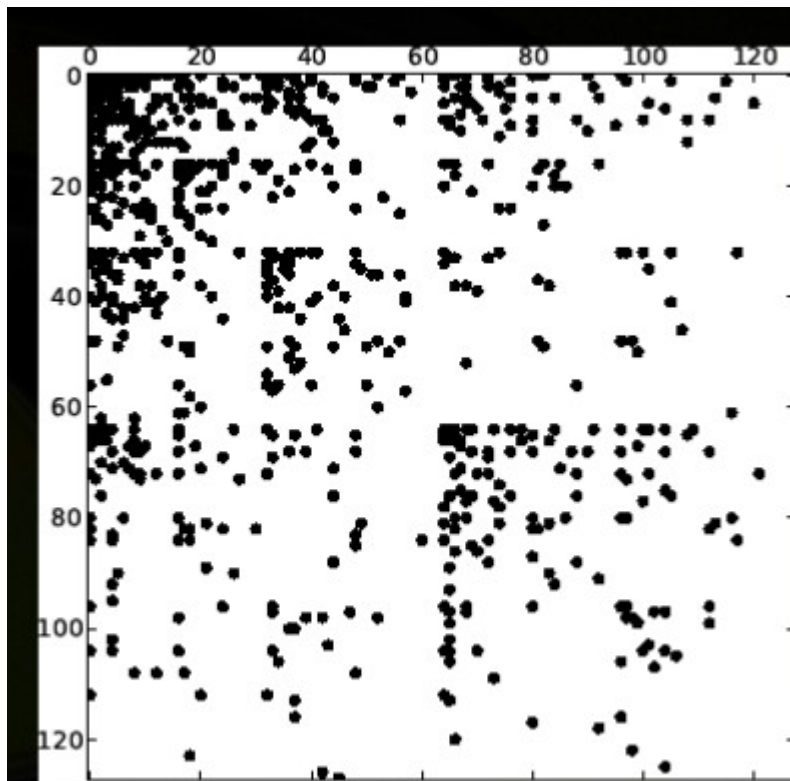
Matrix

Format	Float	Double
COO	12.00	16.00
CSR	8.42	12.42
DIA	76.83	153.65
ELL	14.20	21.29
HYB	9.60	14.20

Bytes per Nonzero Entry

# Que formato elegir?

Power law graph



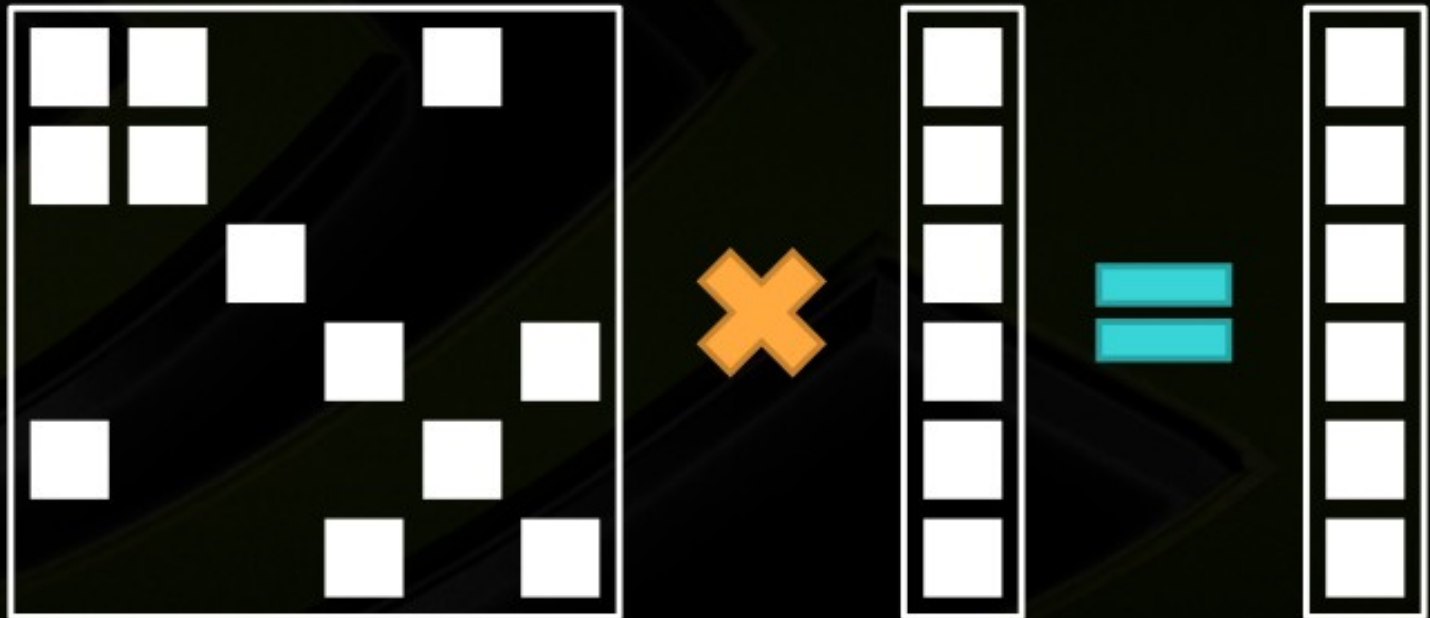
Matrix

Format	Float	Double
COO	12.00	16.00
CSR	8.74	12.73
DIA	118.83	237.66
ELL	49.88	74.82
HYB	13.50	19.46

Bytes per Nonzero Entry

# Sistemas lineales raros

- **Solve sparse linear system ( $A \cdot x = b$ )**
- **Variety of *direct* and *iterative* methods**






# CuSPARSE: Formatos de Matrices Ralas

- Compressed Sparse Row Format (CSR)

The only way the CSR differs from the COO format is that the array containing the row indices is compressed in CSR format. The  $m \times n$  sparse matrix A is represented in CSR format by the following parameters.


1.0	4.0	0.0	0.0	0.0															
0.0	2.0	3.0	0.0	0.0															
5.0	0.0	0.0	7.0	8.0															
0.0	0.0	9.0	0.0	6.0															



`csrValA = 1.0 4.0 2.0 3.0 5.0 7.0 8.0 9.0 6.0`  
`csrRowPtrA = 0 2 4 7 9`  
`csrColIndA = 0 1 1 2 0 3 4 2 4`

<code>nnz</code>	(integer)	The number of nonzero elements in the matrix.
<code>csrValA</code>	(pointer)	Points to the data array of length <code>nnz</code> that holds all nonzero values of A in row-major format.
<code>csrRowPtrA</code>	(pointer)	Points to the integer array of length <code>m+1</code> that holds indices into the arrays <code>csrColIndA</code> and <code>csrValA</code> . The first <code>m</code> entries of this array contain the indices of the first nonzero element in the <code>i</code> th row for <code>i=i,...,m</code> , while the last entry contains <code>nnz+csrRowPtrA(0)</code> . In general, <code>csrRowPtrA(0)</code> is 0 or 1 for zero- and one-based indexing, respectively.
<code>csrColIndA</code>	(pointer)	Points to the integer array of length <code>nnz</code> that contains the column indices of the corresponding elements in array <code>csrValA</code> .

# CuSPARSE: Formatos de Matrices Ralas

- 3.2.2. Coordinate Format (COO)
- 3.2.3. Compressed Sparse Row Format (CSR)
- 3.2.4. Compressed Sparse Column Format (CSC)  Variante que en lugar de usar filas usa columnas
- 3.2.5. Block Compressed Sparse Row Format (BSR)
- 3.2.6. Extended BSR Format (BSRX)

## ▼ 13. cuSPARSE Format Conversion Reference

13.1. <code>cusparse&lt;t&gt;bsr2csr()</code>
13.2. <code>cusparse&lt;t&gt;gebsr2gebsc()</code>
13.3. <code>cusparse&lt;t&gt;gebsr2gebsr()</code>
13.4. <code>cusparse&lt;t&gt;gebsr2csr()</code>
13.5. <code>cusparse&lt;t&gt;csr2gebsr()</code>
13.6. <code>cusparse&lt;t&gt;coo2csr()</code>
13.7. <code>cusparse&lt;t&gt;csc2dense()</code> [DEPRECATED]
13.8. <code>cusparse&lt;t&gt;csr2bsr()</code>
13.9. <code>cusparse&lt;t&gt;csr2coo()</code>
13.10. <code>cusparseCsr2cscEx2()</code>
13.11. <code>cusparse&lt;t&gt;csr2dense()</code> [DEPRECATED]
13.12. <code>cusparse&lt;t&gt;csr2csr_compress()</code>
13.13. <code>cusparse&lt;t&gt;dense2csc()</code> [DEPRECATED]
13.14. <code>cusparse&lt;t&gt;dense2csr()</code> [DEPRECATED]
13.15. <code>cusparse&lt;t&gt;nnz()</code>
13.16. <code>cusparseCreateIdentityPermutation()</code>
13.17. <code>cusparseXcoosort()</code>
13.18. <code>cusparseXcsrsort()</code>
13.19. <code>cusparseXcscsort()</code>
13.20. <code>cusparseXcsru2csr()</code>
13.21. <code>cusparseXpruneDense2csr()</code>
13.22. <code>cusparseXpruneCsr2csr()</code>
13.23. <code>cusparseXpruneDense2csrPercentage()</code>
13.24. <code>cusparseXpruneCsr2csrByPercentage()</code>
13.25. <code>cusparse&lt;t&gt;nnz_compress()</code>

## ▼ 14.3. Sparse Matrix APIs

14.3.1. <code>cusparseCreateCoo()</code>
14.3.2. <code>cusparseCreateCooAoS()</code> [DEPRECATED]
14.3.3. <code>cusparseCreateCsr()</code>
14.3.4. <code>cusparseCreateCsc()</code>
14.3.5. <code>cusparseCreateBlockedEll()</code>
14.3.6. <code>cusparseDestroySpMat()</code>
14.3.7. <code>cusparseCooGet()</code>
14.3.8. <code>cusparseCooAosGet()</code> [DEPRECATED]
14.3.9. <code>cusparseCsrGet()</code>
14.3.10. <code>cusparseCsrSetPointers()</code>
14.3.11. <code>cusparseCscSetPointers()</code>
14.3.12. <code>cusparseBlockedEllGet()</code>
14.3.13. <code>cusparseSpMatGetSize()</code>
14.3.14. <code>cusparseSpMatGetFormat()</code>
14.3.15. <code>cusparseSpMatGetIndexBase()</code>
14.3.16. <code>cusparseSpMatGetValues()</code>
14.3.17. <code>cusparseSpMatSetValues()</code>
14.3.18. <code>cusparseSpMatGetStridedBatch()</code>
14.3.19. <code>cusparseSpMatSetStridedBatch()</code> [DEPRECATED]
14.3.20. <code>cusparseCooSetStridedBatch()</code>
14.3.21. <code>cusparseCsrSetStridedBatch()</code>
14.3.22. <code>cusparseSpMatGetAttribute()</code>
14.3.23. <code>cusparseSpMatSetAttribute()</code>

# Ejericicios CuSPARSE

- **/share/apps/icnpg/clases/Clases\_cusparse/**
  - // Create the cuSPARSE handle
  - // Allocate device memory for vectors and the dense form of the matrix A
  - // Construct a descriptor of the matrix A
  - // Transfer the input vectors and dense matrix A to the device
  - // Compute the number of non-zero elements in A
  - // Allocate device memory to store the sparse CSR representation of A
  - // Convert A from a dense formatting to a CSR formatting, using the GPU
  - // Perform matrix-matrix multiplication with the CSR-formatted matrix A
  - // Copy the result vector back to the host

Si uno hiciera el producto de matrices densas se estarían haciendo muchos productos por cero



# ArrayFire

[Download](#)[Consulting](#)[Training](#)[Blog](#)

## The Fastest Library for GPUs

also, hire us to accelerate your code

Easy-to-use API, like these examples

```
import arrayfire as af
af.set_backend('cuda')
A = af.randu(2**15, 2**15)
A2 = af.matmul(A, A)
B = af.fft2(A2)
```

python

# choose cuda, opencl, or cpu  
# create GPU data  
# fast function calls  
# ...

[Open in Colab \(Python\)](#)

```
#include <arrayfire.h>
auto A = af::randu(2<<15, 2<<15); // create GPU data
auto A2 = af::matmul(A, A); // fast function calls
auto B = af::fft2(A2); // ...
```

c++

[Open in Codespace \(C++\)](#)

Get Started

[Download Binaries](#)

[Github Project](#)

Talk to Us About Your Project

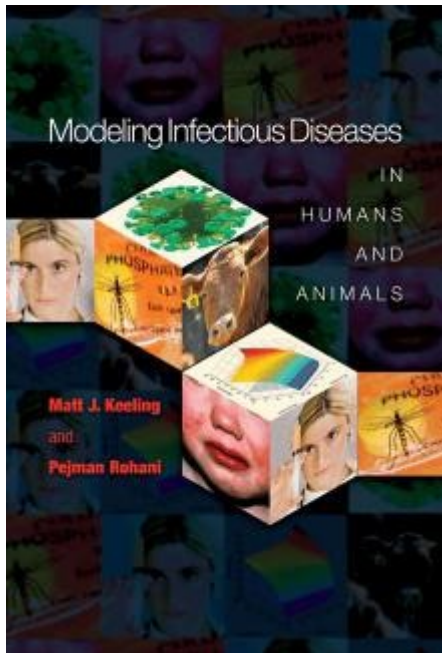
[Learn about Our Services](#)

Mejor performance que cupy, y además corre en CUDA y OPENCL, y se puede usar en C++

(usa cuSOLVER para gpu de nvidia)

# Ejericicios SPARSE con cupy y ArrayFire

- ICNPG\_try-arrayfire-on-colab



$$\frac{dX_i}{dt} = v_i N_i - \lambda_i X_i - \mu_i X_i,$$

$$\frac{dY_i}{dt} = \lambda_i X_i - \gamma_i Y_i - \mu_i Y_i,$$

$$\lambda_i = \beta_i \sum_j \rho_{ij} \frac{X_j}{N_i},$$

↓  
Elementos de matriz  
Típicamente rala

# CuSOLVER

$$Ax = b \quad A_i x_i = b_i$$

$$Ax = \lambda x \quad Ax = \lambda Bx$$

- **¿Qué es?:** A high-level package **based on the cuBLAS and cuSPARSE libraries**... para resolver sistemas de ecuaciones lineales.
- **Operaciones:** matrix factorization and triangular solve routines for dense matrices, a sparse least-squares solver and an eigenvalue solver + a new refactorization library useful for solving sequences of matrices with a shared sparsity pattern.
- **Tres sabores:**
  - CuSolverDN: matrices densas generales.
  - CuSolverSP: formato CSR, matrices generales o especiales (simétricas/hermíticas, etc).
  - CuSolverRF: refactorización, útil para sistemas “batcheados”

cuSOLVER
▷ 1. Introduction
▷ 2. Using the CUSOLVER API
▷ 3. Using the CUSOLVERMG API
▷ A. cuSolverRF Examples
▷ B. CSR QR Batch Examples
▷ C. QR Examples
▷ D. LU Examples
▷ E. Cholesky Examples
▷ F. Examples of Dense Eigenvalue Solver
▷ G. Examples of Singular Value Decomposition
▷ H. Examples of multiGPU eigenvalue solver
▷ I. Examples of multiGPU linear solver
J. Acknowledgements
K. Bibliography

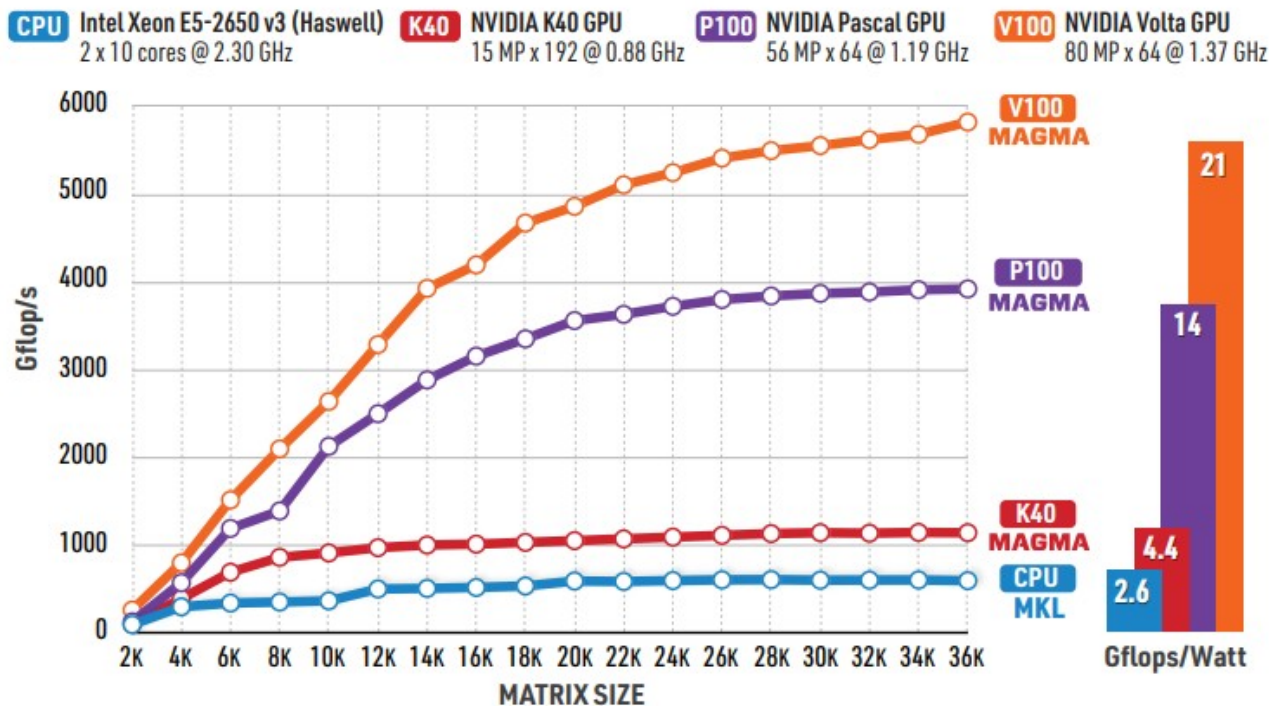
# MAGMA

# MAGMA

- Algebra lineal numérica para sistemas híbridos/heterogéneos: CPU-multicore + GPUs + ?

## PERFORMANCE & ENERGY EFFICIENCY

### MAGMA LU factorization in double precision arithmetic



CUDA	OpenCL	Intel Xeon Phi	Linear system solvers
			Eigenvalue problem solvers
			Auxiliary BLAS
			Batched LA
			Sparse LA
			CPU/GPU Interface
			Multiple precision support
			Non-GPU-resident factorizations
			Multicore and multi-GPU support
NEW			MAGMA Analytics/DNN
			LAPACK testing
			Linux
			Windows
			Mac OS

<https://icl.utk.edu/magma/overview/index.html>

<http://www.icl.utk.edu/files/print/2017/magma-sc17.pdf>

(usa cuSOLVER para gpu de nvidia)

# Ejercicios CuSolver

- `/share/apps/icnpg/clases/Clases_cusolver/`

# Linear algebra (`cupy.linalg`)

**Hint**

NumPy API Reference: Linear algebra (`numpy.linalg`)

**See also**

Linear algebra (`cupyx.scipy.linalg`)

# Linear algebra (`cupyx.scipy.linalg`)

**Hint**

SciPy API Reference: Linear algebra (`scipy.linalg`)

## Linear algebra ( `cupyx.scipy.linalg` )

`cupyx.scipy.linalg.solve_triangular`  
`cupyx.scipy.linalg.tril`  
`cupyx.scipy.linalg.triu`  
`cupyx.scipy.linalg.lu`  
`cupyx.scipy.linalg.lu_factor`  
`cupyx.scipy.linalg.lu_solve`  
`cupyx.scipy.linalg.block_diag`  
`cupyx.scipy.linalg.circulant`  
`cupyx.scipy.linalg.companion`  
`cupyx.scipy.linalg.convolution_matrix`  
`cupyx.scipy.linalg.dft`  
`cupyx.scipy.linalg.fiedler`  
`cupyx.scipy.linalg.fiedler_companion`  
`cupyx.scipy.linalg.hadamard`  
`cupyx.scipy.linalg.hankel`  
`cupyx.scipy.linalg.helmert`  
`cupyx.scipy.linalg.hilbert`  
`cupyx.scipy.linalg.kron`  
`cupyx.scipy.linalg.leslie`  
`cupyx.scipy.linalg.toeplitz`  
`cupyx.scipy.linalg.tri`

## Sparse matrices ( `cupyx.scipy.sparse` )

`cupyx.scipy.sparse.coo_matrix`  
`cupyx.scipy.sparse.csc_matrix`  
`cupyx.scipy.sparse.csr_matrix`  
`cupyx.scipy.sparse.dia_matrix`  
`cupyx.scipy.sparse.spmatrix`  
`cupyx.scipy.sparse.eye`  
`cupyx.scipy.sparse.identity`  
`cupyx.scipy.sparse.kron`  
`cupyx.scipy.sparse.diags`  
`cupyx.scipy.sparse.spdiags`  
`cupyx.scipy.sparse.tril`  
`cupyx.scipy.sparse.triu`  
`cupyx.scipy.sparse.bmat`  
`cupyx.scipy.sparse.hstack`  
`cupyx.scipy.sparse.vstack`  
`cupyx.scipy.sparse.rand`  
`cupyx.scipy.sparse.random`  
`cupyx.scipy.sparse.find`  
`cupyx.scipy.sparse.issparse`  
`cupyx.scipy.sparse.isspmatrix`  
`cupyx.scipy.sparse.isspmatrix_csc`  
`cupyx.scipy.sparse.isspmatrix_csr`  
`cupyx.scipy.sparse.isspmatrix_coo`  
`cupyx.scipy.sparse.isspmatrix_dia`

## Linear algebra ( `cupy.linalg` )

`cupy.dot`  
`cupy.vdot`  
`cupy.inner`  
`cupy.outer`  
`cupy.matmul`  
`cupy.tensordot`  
`cupy.einsum`  
`cupy.linalg.matrix_power`  
`cupy.kron`  
`cupy.linalg.cholesky`  
`cupy.linalg.qr`  
`cupy.linalg.svd`  
`cupy.linalg.eigh`  
`cupy.linalg.eigvalsh`  
`cupy.linalg.norm`  
`cupy.linalg.det`  
`cupy.linalg.matrix_rank`  
`cupy.linalg.slogdet`  
`cupy.trace`  
`cupy.linalg.solve`  
`cupy.linalg.tensorsolve`  
`cupy.linalg.lstsq`  
`cupy.linalg.inv`  
`cupy.linalg.pinv`

CUPY



# CUPY

## Linear Algebra

Matrix and vector products

Decompositions

Matrix eigenvalues

Norms etc.

Solving linear equations

Logic Functions

Mathematical Functions

Padding

Random Sampling (`cupy.random`)

Sorting, Searching, and Counting

Statistical Functions

CuPy-specific Functions

SciPy-compatible Routines

Sparse matrices

Multi-dimensional image processing

NumPy-CuPy Generic Code Support

Memory Management

Low-Level CUDA Support

[Docs](#) » [Reference Manual](#) » [Routines](#) » Linear Algebra

[Edit on GitHub](#)

## Linear Algebra

### Matrix and vector products

<code>cupy.cross</code>	Returns the cross product of two vectors.
<code>cupy.dot</code>	Returns a dot product of two arrays.
<code>cupy.vdot</code>	Returns the dot product of two vectors.
<code>cupy.inner</code>	Returns the inner product of two arrays.
<code>cupy.outer</code>	Returns the outer product of two vectors.
<code>cupy.matmul</code>	Returns the matrix product of two arrays and is the implementation
<code>cupy.tensordot</code>	Returns the tensor dot product of two arrays along specified axes.
<code>cupy.einsum</code>	Evaluates the Einstein summation convention on the operands.
<code>cupy.linalg.matrix_power</code>	Raise a square matrix to the (integer) power $n$ .
<code>cupy.kron</code>	Returns the kronecker product of two arrays.

# Matrix computations on the GPU

CUBLAS, CUSOLVER and MAGMA by example

**Andrzej Chrzęszczyk**

*Jan Kochanowski University, Kielce, Poland*

**Jacob Anders**

*CSIRO, Canberra, Australia*

Version 2017

**Matrix Computations  
on GPU  
with ArrayFire - Python  
and ArrayFire - C/C++**

Andrzej Chrzęszczuk

Jan Kochanowski University

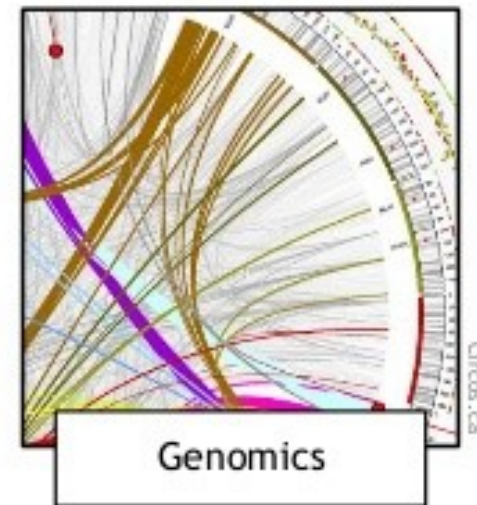
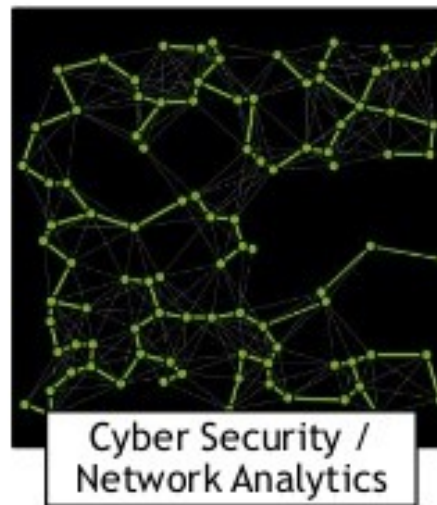
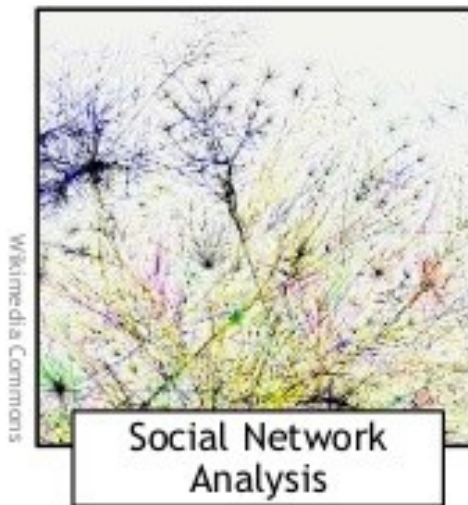
Version 2017

# NVGRAPH (C) / CUGRAPH (python)

- graph construction and manipulation primitives, and a set of useful graph algorithms optimized for the GPU.

## GRAPH ANALYTICS

Insight from Connections in Big Data



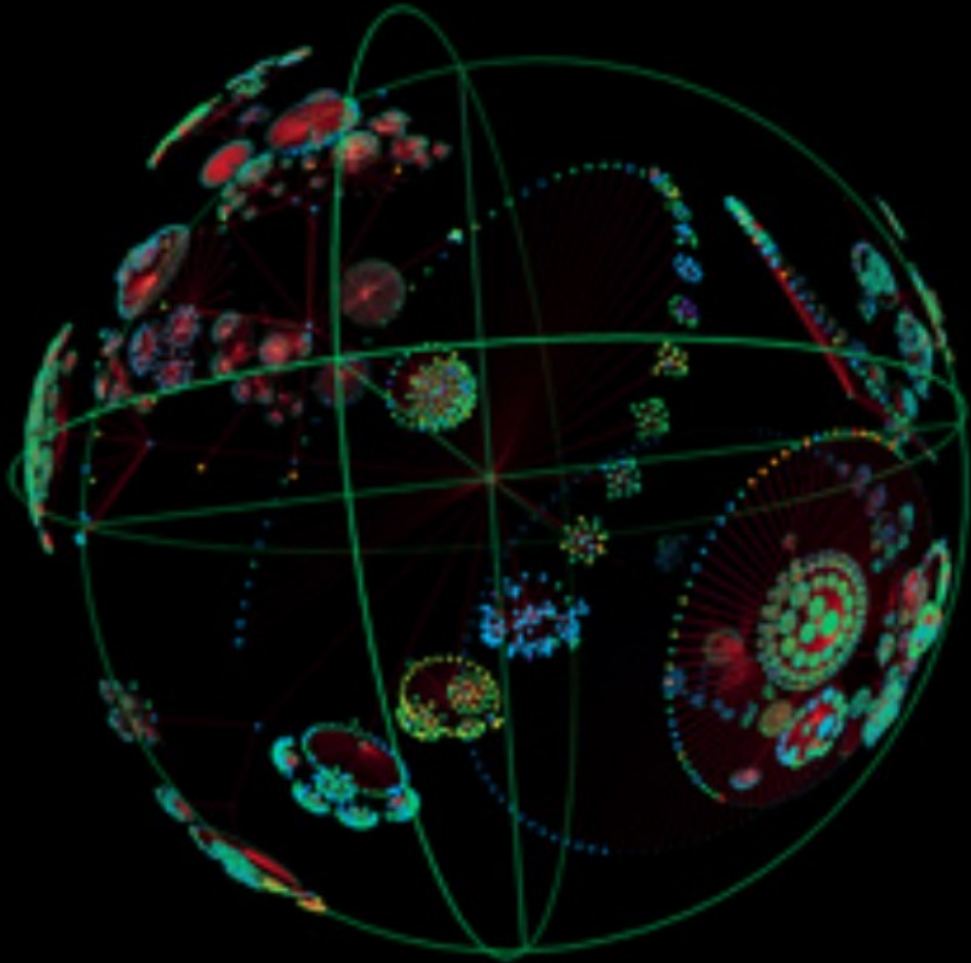
... and much more: Parallel Computing, Recommender Systems, Fraud Detection, Voice Recognition, Text Understanding, Search





# Dynamical Processes on Complex Networks

Alain Barrat, Marc Barthélemy, Alessandro Vespignani



CAMBRIDGE

Copyrighted Material

Albert-László  
Barabási

L I N K E D

The New Science  
of Networks

How Everything is Connected to Everything Else  
and What it Means for Science, Business  
and Everyday Life

Copyrighted Material

# RAPIDS (python or C++)

## DATA SCIENCE AT THE SPEED OF THOUGHT

### ▶ WHAT IS RAPIDS

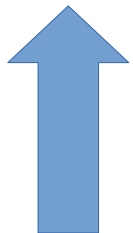
RAPIDS provides unmatched speed with familiar APIs that match the most popular PyData libraries. Built on the shoulders of giants including **NVIDIA CUDA** and **Apache Arrow**, it unlocks the speed of GPUs with code you already know.

[Learn more on the About Section ↗](#)

### 📦 FASTER PANDAS WITH CUDF

cuDF is a near drop in replacement to pandas for most use cases and has greatly improved performance.

[Run this benchmark yourself ↗](#)



### ⚡ WHY USE RAPIDS

RAPIDS allows fluid, creative interaction with data for everyone from BI users to AI researchers on the cutting edge. GPU acceleration means less time and less cost moving data and training models.

[Find out more from RAPIDS Use Cases ↗](#)

### 🧠 FASTER SCIKIT-LEARN WITH CUML

cuML brings huge speedups to ML modeling with an API that matches scikit-learn.

[Run this benchmark yourself ↗](#)

### 🔗 OPEN SOURCE ECOSYSTEM

RAPIDS is Open Source and available on [GitHub](#). Our mission is to empower and advance the open-source GPU data science data engineering ecosystem.

[Jump to RAPIDS on GitHub ↗](#)

### 🔗 FASTER NETWORKX WITH CUGRAPH

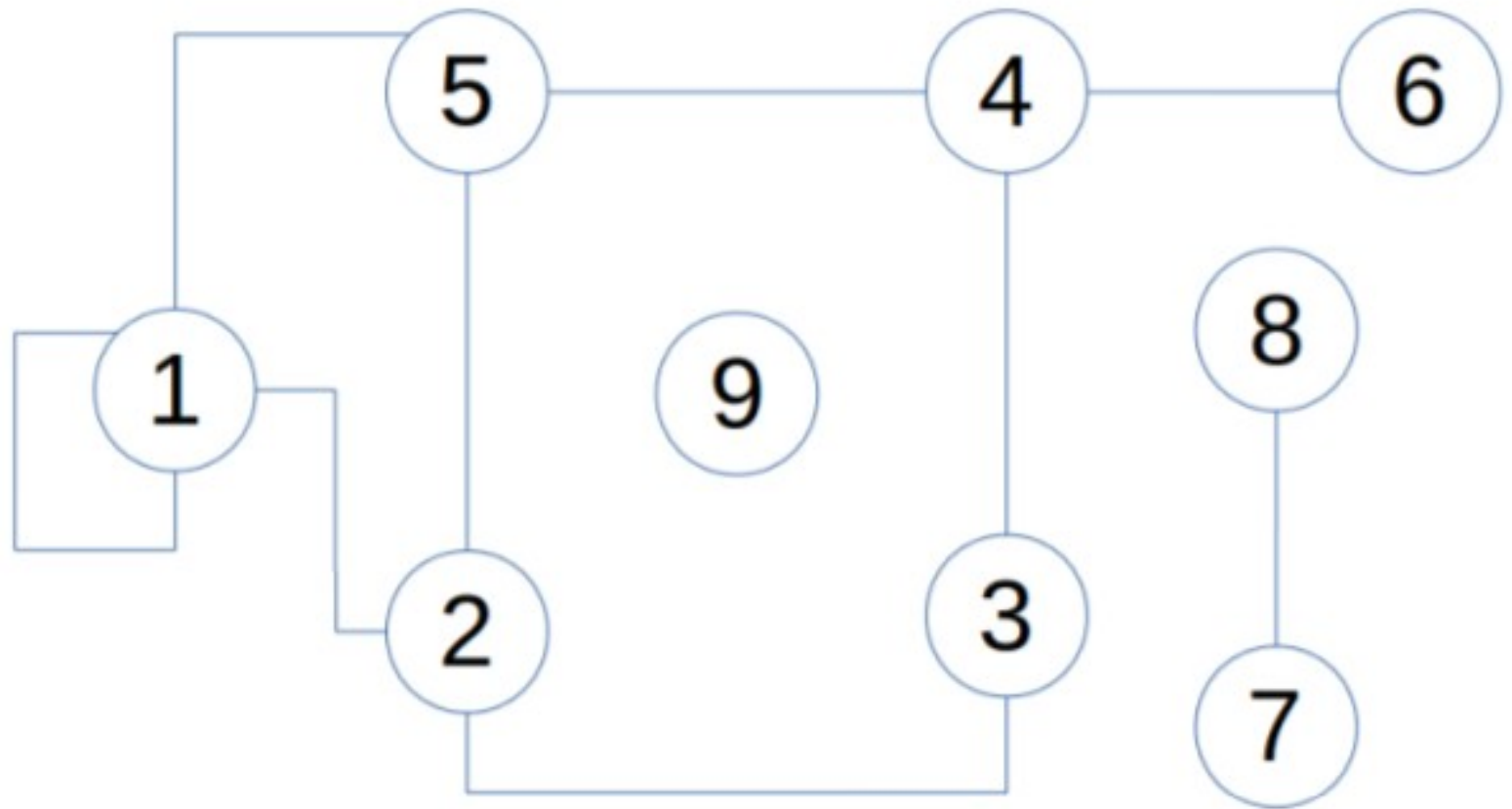
cuGraph makes migration from networkX easy, accelerates graph analytics, and allows scaling far beyond existing tools.

[Run this benchmark yourself ↗](#)





# Ejercicio



Usando CUGRAPH

- Leer matriz de adyacencias csv.
- Número de caminos de exactamente 5 pasos que unen un vértice con cualquier otro.
- Número de componentes del grafo, y sus vértices miembros.
- Colorear, como si fuera un mapa, los vértices usando la mínima cantidad de colores.