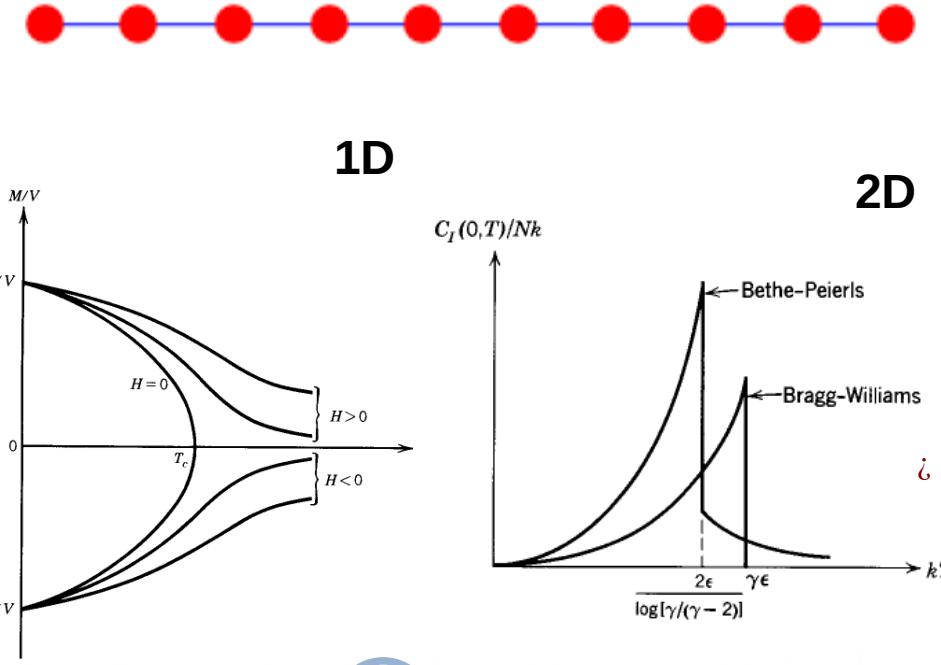
Modelo de Ising



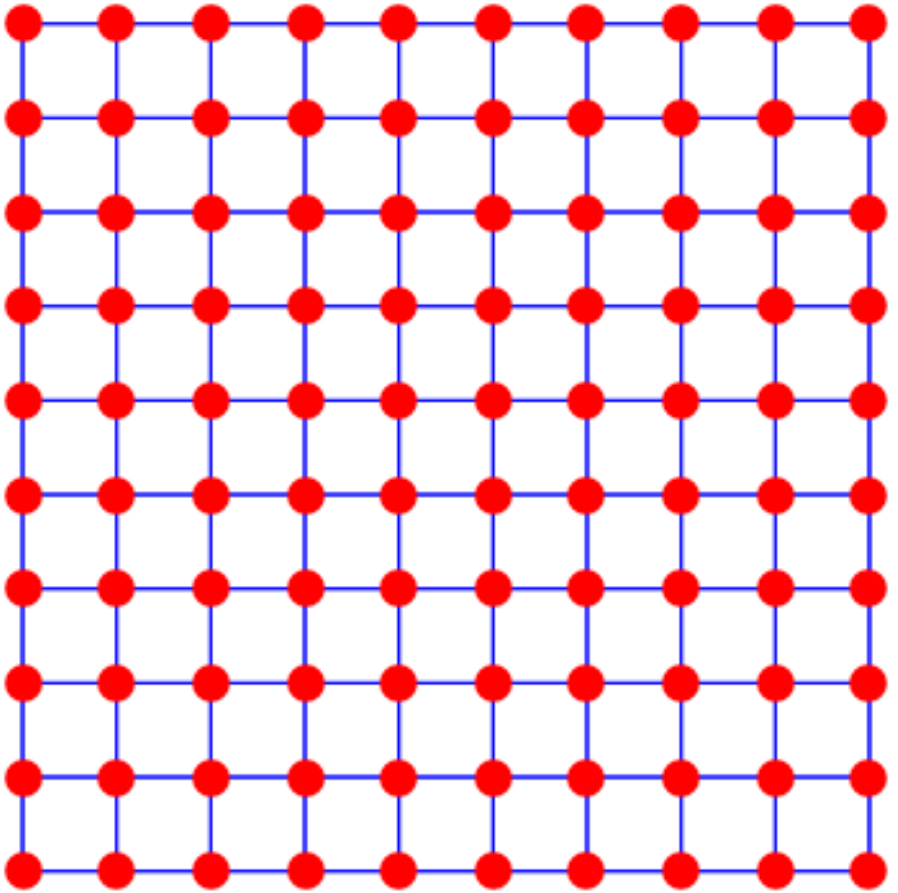
1D

2D

$$-J \sum_{<i,j>} S_z^i S_z^j - \mu B \sum_i S_z^i$$

Interacción a primeros vecinos

Campo externo



Dos estados, ↑ y ↓, para cada sitio. Entonces, 2^{Ns} configuraciones.

Ensamble canónico

$$Z_C = \sum_{\substack{\text{estados} \\ \text{L}\times\text{L espines}}} e^{-\beta E_n}$$

$$U = \frac{1}{Z_C} \sum_{\substack{\text{estados} \\ \text{L}\times\text{L espines}}} E_n \, e^{-\beta E_n}$$

$$U = \sum_{\substack{\text{estados} \\ \text{L}\times\text{L espines}}} E_n \, p_n$$

$$p_n = \frac{1}{Z_C} e^{-\beta E_n}$$

Probabilidad

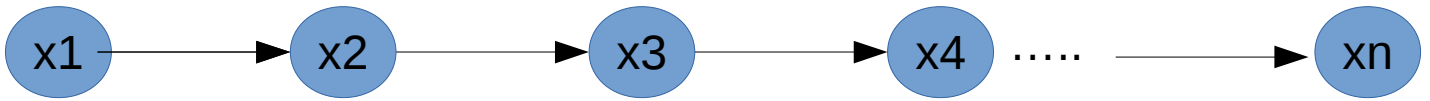
Idea: Si tengo configuraciones (o sea estados!) que tengan la distribucion de probabilidad adecuada

$$\langle O \rangle = \sum_n p_n \langle n|O|n\rangle \simeq \frac{1}{n_\alpha} \sum_{\substack{n_\alpha \text{ estados} \\ \text{representativos}}} \langle \alpha|O|\alpha\rangle$$

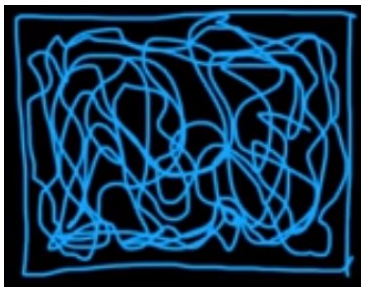
SAMPLING

Metropolis Monte Carlo

- Monte Carlo es un nombre genérico para hacer sampling.
- En los métodos Monte Carlo de cadena de Markov, como Metropolis!. Un estado sigue al otro, y depende solo del anterior. Las transiciones se contruyen de manera que en el estado estacionario (luego de muchas repeticiones) las configuraciones provienen de la distribución deseada p_n



P(x4|x3,x2,x1) = P(x4|x3)



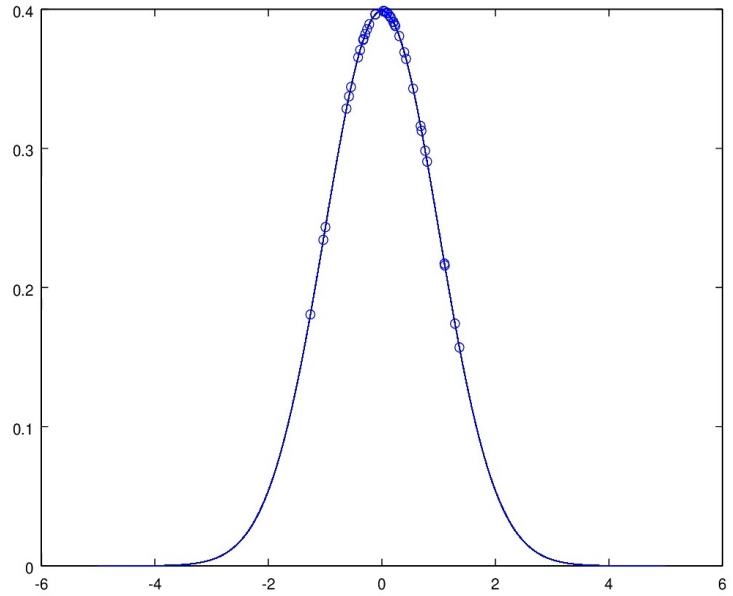
Ergódico

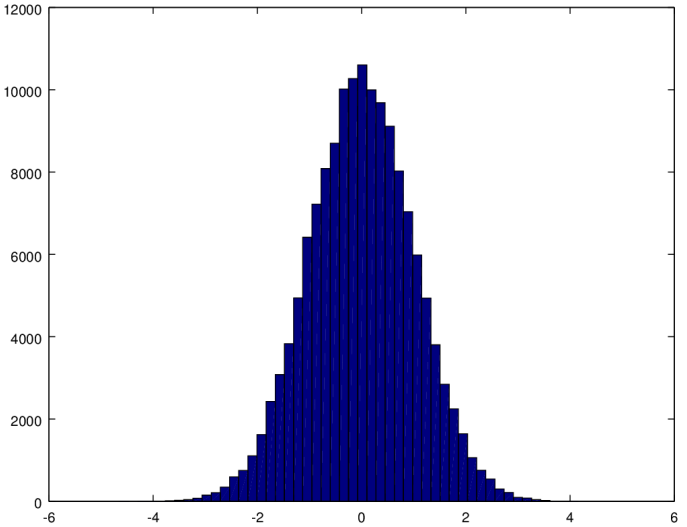
Algoritmo de Metropolis

1. Elijo un estado posible *x*, para iniciar la cadena
2. Ciclo
   i. Genero un estado distinto *x′* (con alguna dist.)
   ii. Calculo *a* = Min{ 1, p(x')/p(x)}
   iii. Con probabilidad *a* acepto el cambio $x \rightarrow x'$
   iv. Guardo x' (si acepté) o repito el mismo x sino.

NB: p(x) no necesita estar normalizada!!!

Ejemplo con distribución Gaussiana

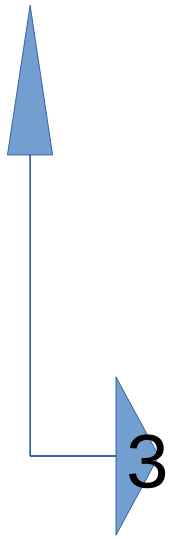




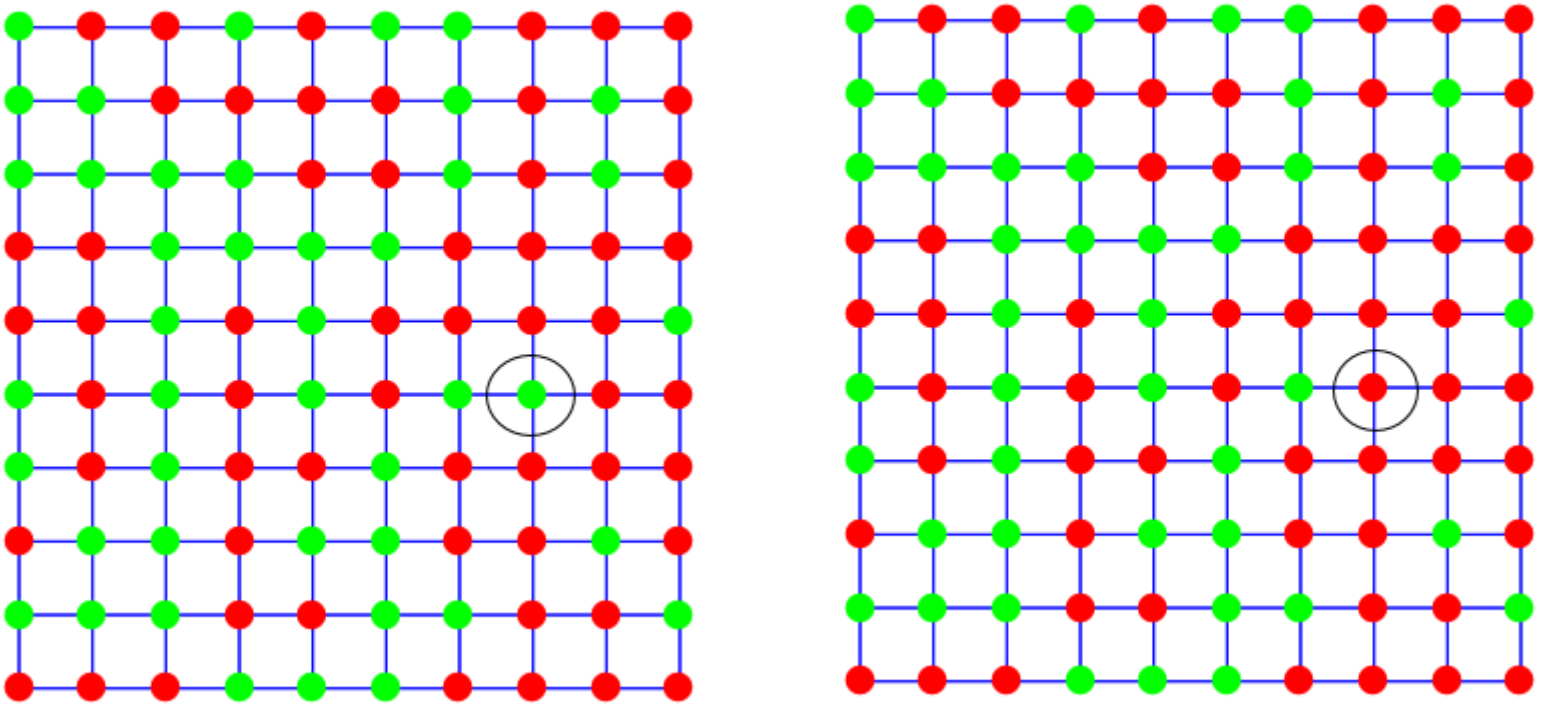

```
x(1)=0.5; %primer paso de mi cadena de Markov.

%propongo un cambio

for i=1:npasos-1;
x_nuevo = 10*(rand()-0.5);

a = min(1,exp(-x_nuevo^2/2)/exp(-x(i)^2/2));
%notar que nunca haria un cociente asi de exps, sino poner el argumento

if(rand() < a) % acepto con probalidad a
        x(i+1) = x_nuevo;
        aceptados = aceptados + 1;
else
        x(i+1) = x(i);

end

end
```

Sampling ising: aceptamos y rechazamos

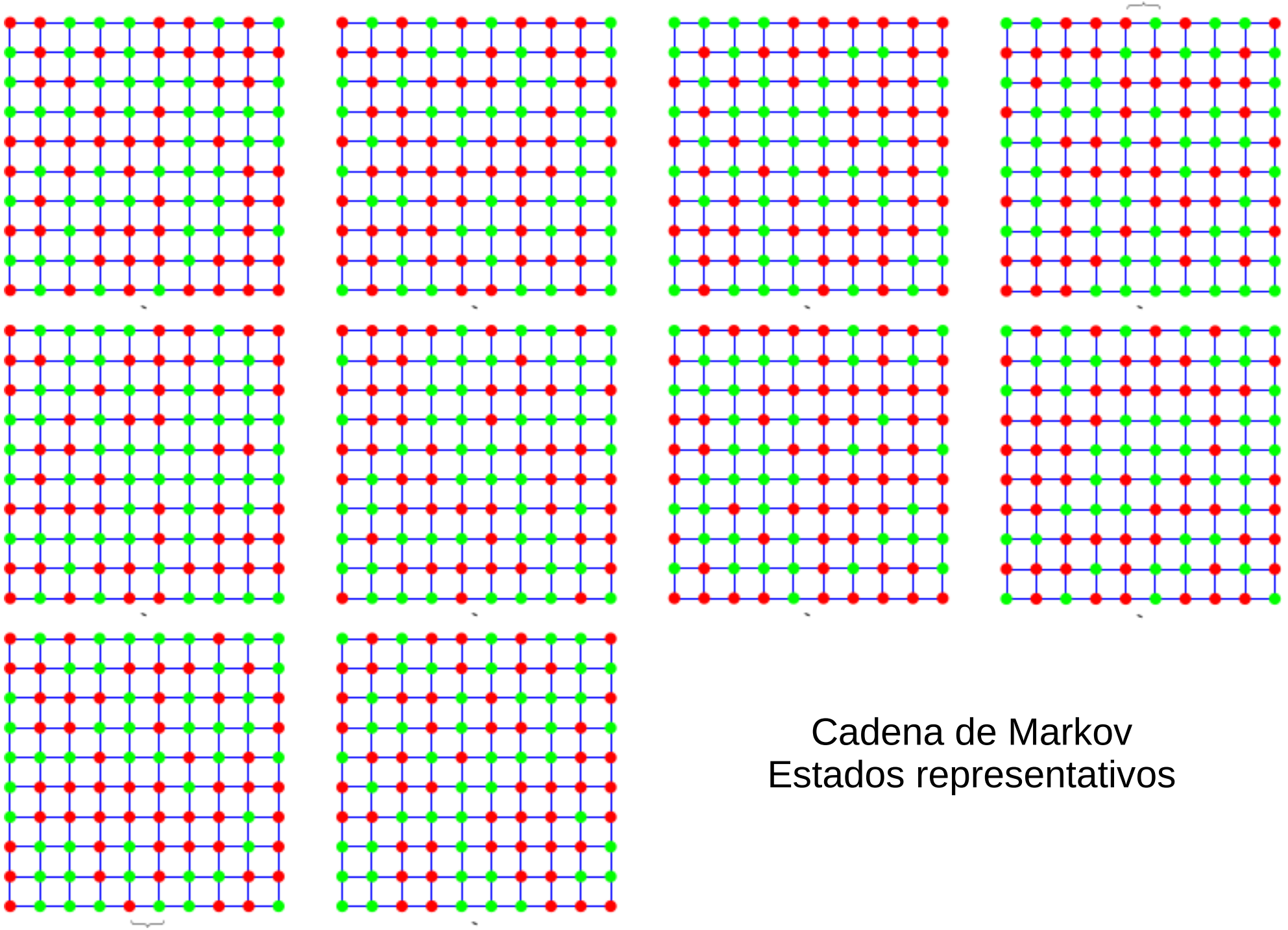
1. Partimos de una configuracion inicial (como elegirla?)
2. Proponemos un cambio de un espin de la red (flip)
   i. La energía baja? Genial lo aceptamos.
   ii. La energía sube? Lo aceptamos con probabilidad $e^{-\beta \Delta E}$
   iii. Repetimos para todos los espines de la red.
3. Volvemos a recorrer toda la red
4. Para los estados de la cadena, calculamos los valores de las magnitudes que estemos interesados.



Ejemplo



Flip



Cadena de Markov
Estados representativos

Como lo hacemos?

- Si sabes programar en C, C++, Fortran, Python o algun otro lenguaje, mejor es más rapido.
- Si no, (y no queres aprender..) podes usar entornos como matlab, octave, etc.
- Luego para presentar los resultados usas lo que preferis (word+origin, latex+octave/grace)
- Aca vamos a dar una brevisima introducción a Octave (o matlab).

Octave / Matlab

Algunas ayudas básicas. (el comando help es tu amigo)

- Definimos variables (matrices en general)
- Las mayúsculas y minúsculas son distintas, e.g. A ≠ a .
- Un ; al final de la intrucción hace que no se muestre el resultado (pero se calcula!).
- Rangos: e.g., vector a = (0, 0.1, 0.2, ….1)
  a = 0:0.1:1; genera un vector desde 0 a 1 con paso 0.1; Es lo mismo que decir a=[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
- Si A es una matriz de 2x2, los elementos son A(1,1), A(1,2), A(2,1),A(2,2)
- Un bucle/ciclo (do o for en otros lenguajes) y un condicional tienen una estructura comun.

```
for i = 1:10
  Hago cosas..
end
```

```
if (condicion)
  Hago cosas...
end
```

Operadores:
== igual
~= distinto
>, < , >=, <=

- Un conjunto de instrucciones se pueden poner en un archivo tal cual como lo ejecutaria, eso es un script, el archivo debe llamarse “micalculo.m” y se ejecutan escribiendo “micalculo” dentro de octave/Matlab.
- En los scripts puedo poner comentarios que no se ejecutan, para eso utilizo el **%**
- Tambien puedo definir funciones en archivos separados .m,

```
function [out1,out2]=mifuncion(a,b)
Hago cosas  y asigno out1 y out2
end %no es necesario siempre
```

Probemos Octave viendo los generadores de numeros aleatorios