# Practical session 2

**Authors Ferran Dalmau Codina: u150401 206744 and Jalal Douiri: U150388 206338**[1]

[*] *Universitat Pompeu Fabra*

*Compiled May 7, 2019*

## 1. GRADIENT DESCEND

## 2. EXERCISE 1

toy_fun.m

```
y = (1/500)*((x(1)^4)+(x(2)^4)-(90*x(1)^2)
    -(60*x(2)^2)+(100*x(1))+(20*x(2))+1);
```

toy_gradient.m

```
grad = zeros(2,1);
% >>> complete >>>
grad(1) = 1/500*(4*x(1)^3-180*x(1)+100);
grad(2) = 1/500*(4*x(2)^3-120*x(2)+20);
```

La variable grad(1) almazena la derivada parcial respecto x(1).
La variable grad(2) almazena la derivada parcial respecto x(2).

## 3. EXERCISE 2

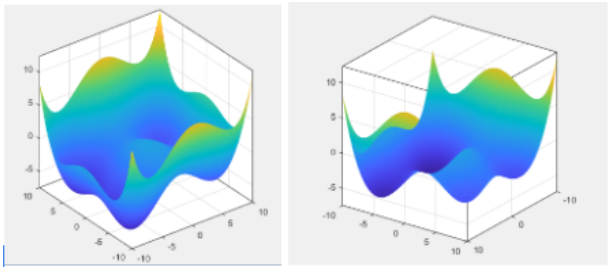
gradient_descent.m

```
norm_gf=inf;
iter=0;
while (iter <= max_iters)
    iter=iter+1;
        % keep x_old - just for visualization
        x_old = x;
        % gradient descent update
    gf=grad(x_old,fun_prms);
    x =x_old - step_size*gf;
```

Grad es un puntero a la funcion que retorna el gradiente de la funcion correspondiente, y fun_params es una classe que contine informacion relativa a step size y numero de iteraciones. **x = x_old - step_size*gf** actualiza el valor de x de manera que el coste de la funcion se minimiza.

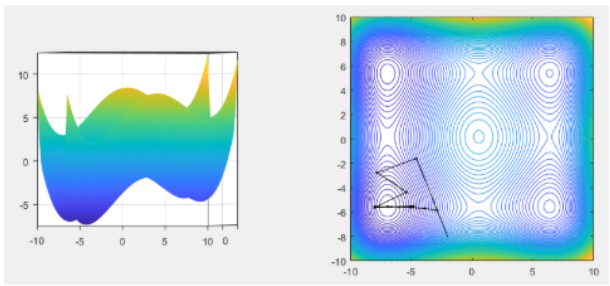
## 4. EXERCISE 3

**A. How many local minima does the function have in the domain**



Como podemos observar en las capturas anteriores, deducimos que en el dominio [10, 10]×[10, 10] hay quatro mínimos, tres de ellos son mínimos locales y uno mínimo local.

**B. Run the gradient descent starting from . Does it converge to the global minimum?**



Cuando cambiamos el punto de inicio del gradient descent, podemos ver que ahora tiende a un mínimo global ya que es el punto más bajo que podemos apreciar de la gráfica.

### C. How many iterations are needed to converge?

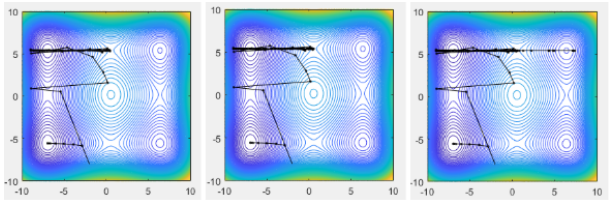


**Fig. 1.** Para un step_size = 4 y iteraciones de 100, 200 y 300 respectivamente

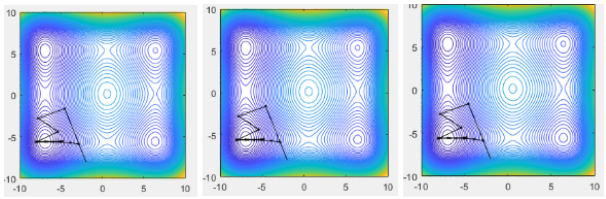


**Fig. 2.** Para un step_size = 3 y iteraciones de 100, 200 y 300 respectivamente

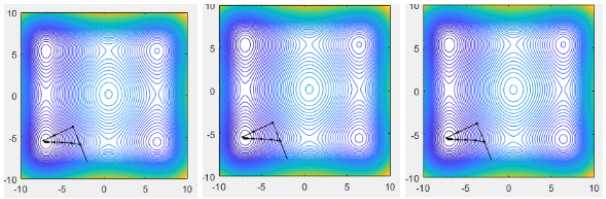
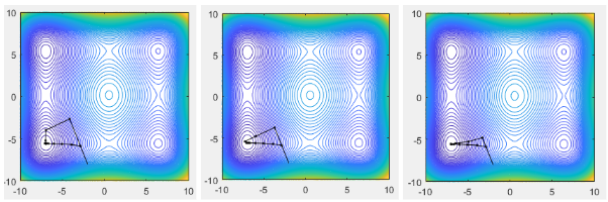
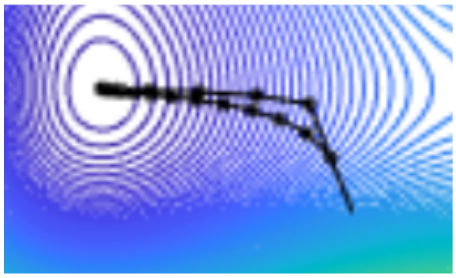


**Fig. 3.** Para un step_size = 2 y iteraciones de 100, 200 y 300 respectivamente

Como podemos ver en las siguientes imágenes, el gradiente solo cambia si cambiamos el tamaño del paso (step_size), también podemos ver que cada vez que disminuimos el tamaño del paso, converge más.

### D. Try different step sizes. Which step sizes yield a faster convergence? Which are more accurat?





En las imágenes anteriores podemos ver que cuando más pequeño el step size, mas converge. El paso que converge más, es el que se aproxima más al 0.
Probamos el step_size=0.5 y vemos que se aproxima bastante.

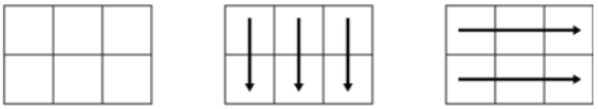
## 5. IMAGE DENOISING ENERGY

### A. Exercise 4

im_fwd_gradient.m

```
function [gradu_j,gradu_i] = im_fwd_gradient(u)
gradu_j = diff(u,1,2);
gradu_j(:, end+1) = 0;
gradu_j(end,:) = 0;
gradu_i = diff(u,1,1);
gradu_i(:, end) = 0;
gradu_i(end+1,:) = 0;
```

Para computar el **forward gradient** de cada eje hacemos uso de la funcion **diff()**,el forward graident en el eje x (diff(A,1,2)) realiza la operacion $u_(i+1,j) - u_(i,j)$ mientras que (diff(A,1,1)) realiza el forward gradient en el eje y $u_(i,j+1) - u_(i,j)$
En este caso las condiciones de frontera usadas son las condiciones de **Newman**, aunque existen diversas metodologias como **Dirichlet**. En newman el gradiente de las conidicones de frontera son 0, por esta razon se añaden 0 a la ultima columna e fila en los dos gradientes.




im_bwd_divergence.m

```
function divg = im_bwd_divergence(gj,gi)
a = [gj(1,:); gj(1:end,:)];
gradu_j = diff(a,1,1);
b = [gi(:,1), gi];
gradu_i = diff(b,1,2);
divg = gradu_j + gradu_i;
```

Esta función computa el **backward divergence** de gi, gj, que tomaran el valor de los gradientes parciales multiplicados por una matriz c que se mantendrà constante.
Para poder realizar el backward gradient,tenemos que añadir una fila extra en caso de **diff(A,1,1)**, e una columna extra en el caso de **diff(A,1,2)**.
Primero duplicamos la primera fila en la matriz gj, para crear una condicion de frontera tipo **Newman** cuando se aplique el gradiente diff(A,1,1).

a | gradu_j

513x512 double

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 25.8281 | 5.2518 | 26.2847 | -3.2362 | 6.1369 | -13.9099 | -1.3069 | 0.7083 | 28.8547 | -34.2703 | 20.9614 |
| 2 | 25.8281 | 5.2518 | 26.2847 | -3.2362 | 6.1369 | -13.9099 | -1.3069 | 0.7083 | 28.8547 | -34.2703 | 20.9614 |
| 3 | -81.2140 | -0.9873 | 33.7474 | -44.6504 | 47.8625 | -24.0018 | -3.7579 | -59.2135 | 0.8399 | 24.1062 | -34.1043 |
| 4 | 61.3898 | 14.8232 | -75.3786 | -6.9736 | -54.4170 | 14.2468 | -26.8958 | 39.7142 | 22.3961 | -39.9471 | 21.9578 |
| 5 | -10.6377 | -2.9548 | 28.2874 | -39.8772 | 27.2099 | 12.2940 | -11.1987 | -17.7579 | -58.3832 | 26.6391 | 1.5750 |
| 6 | -30.0072 | 9.0281 | -35.6158 | 7.6366 | -3.4301 | 21.0559 | 33.4783 | 8.6156 | 18.3108 | 17.1982 | 19.7900 |
| 7 | 13.3587 | 16.3173 | 59.9120 | 2.2480 | 14.0342 | -41.3536 | 16.3582 | 39.2052 | 15.4278 | -33.6823 | -18.2237 |
| 8 | 13.3858 | -12.9097 | -5.7525 | 20.6340 | -22.6921 | 42.3611 | -56.2799 | -47.3124 | -11.5047 | 13.4076 | -5.9721 |
| 9 | 61.4860 | -52.6678 | -58.7192 | -0.0458 | -57.7524 | 30.9480 | -17.1575 | -17.9482 | 47.2285 | 27.5137 | -29.1128 |
| 10 | -17.1166 | -21.7180 | 71.2417 | -6.4301 | 26.6538 | -70.2484 | 69.5035 | 36.9166 | -49.5443 | -51.6704 | -5.5344 |
| 11 | -81.0754 | 65.0175 | -61.6089 | -6.9001 | 28.3940 | 18.8632 | -29.7906 | -10.9458 | -43.7749 | -2.4990 | 43.1156 |
| 12 | 88.2838 | -53.2350 | 26.7531 | -13.5473 | -2.9686 | -20.9460 | 20.6396 | -9.7192 | 62.7192 | 40.7273 | -25.7967 |

**Fig. 4.** matriz a con condiciones de Newman aplicadas

a | gradu_j

512x512 double

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | -107.0421 | -6.2391 | 7.4627 | -41.4142 | 41.7256 | -10.0919 | -2.4510 | -59.9218 | -28.0148 | 58.3765 | -55.0656 |
| 3 | 142.6037 | 15.8105 | -109.1260 | 37.6768 | -102.2795 | 38.2486 | -23.1379 | 98.9277 | 21.5562 | -64.0534 | 56.0621 |
| 4 | -72.0275 | -17.7780 | 103.6660 | -32.9037 | 81.6270 | -1.9528 | 15.6971 | -57.4720 | -80.7793 | 66.5862 | -20.3828 |
| 5 | -19.3694 | 11.9829 | -63.9032 | 47.5139 | -30.6400 | 8.7620 | 44.6770 | 26.3735 | 76.6940 | -9.4409 | 18.2150 |
| 6 | 43.3658 | 7.2892 | 95.5278 | -5.3887 | 17.4643 | -62.4096 | -17.1200 | 30.5896 | -2.8830 | -50.8804 | -38.0137 |
| 7 | 0.0271 | -29.2270 | -65.6644 | 18.3860 | -36.7262 | 83.7147 | -72.6382 | -86.5176 | -26.9325 | 47.0899 | 12.2516 |
| 8 | 48.1002 | -39.7581 | -52.9667 | -20.6797 | -35.0603 | -11.4131 | 39.1225 | 29.3642 | 58.7332 | 14.1061 | -23.1407 |
| 9 | -78.6026 | 30.9497 | 129.9609 | -6.3844 | 84.4062 | -101.1963 | 86.6610 | 54.8648 | -96.7728 | -79.1841 | 23.5784 |
| 10 | -63.9588 | 86.7355 | -132.8506 | -0.4699 | 1.7402 | 89.1116 | -99.2941 | -47.8624 | 5.7694 | 49.1713 | 48.6500 |
| 11 | 169.3592 | -118.2525 | 88.3620 | -6.6472 | -31.3627 | -39.8092 | 50.4303 | 1.2266 | 106.4941 | 43.2264 | -68.9123 |
| 12 | -133.2866 | 91.9259 | -8.6958 | 28.9298 | 33.7578 | -2.4649 | -16.5242 | 42.2990 | -104.2172 | -61.8284 | 34.9544 |

**Fig. 5.** matriz a despues de aplicar diff(a,1,1)

### B. Exercise 5

denoise_energy_gradient.m

```
function grade = denoise_energy_gradient(u,prms)
f    = prms.f;
c    = prms.c;
beta = prms.beta;
%%TO COMPLETE
[gj,gi] = im_fwd_gradient(u);
x = im_bwd_divergence(c.*gi,c.*gj);
grade = (-2*(x))+(2*beta*(u-f));
```

En este codigo el gradiente de la energia que es usado en **Gradient Descend** viene dado por **grade**, donde **beta** es un escalar que tiene el papel de mantener similitudes entre la imagen original y la denoised, a mayor beta, menor es el cambio respecto la imagen original. Este cambio se manifiesta de distintas formas dependiendo de la matriz **f** que que magnifica la energia de los pixeles que conforman edges en la imagen, pero a grandes rasgos una beta pequeña implica una imagen con mas **blur.**

### C. Exercise 6: Results

**Fig. 6.** Original image without noise

**Fig. 7.** Original image with noise

En los dos ejemplos el gradient step se ha mantenido constante en .09 con 2000 iteraciones y la matriz c ha sido generada por la función **compute edge map.**

**Fig. 8.** matriz c

Como podemos observar los pixeles que conforman los edges (altas frequencias )toman valores cercanos al 0 mientras que los pixeles que estan en zonas de bajas frequencias toman valores cercanos al uno (como podemos observar la imagen es similar a implementar un sobel filter normalizado). Con este filtro conseguimos que los edges de la imagen tienden a preservarse mientras las partes de baja frequencia se modifiquen con mas facilidad porque to tienen tanto impacto en terminos de energia.

**Fig. 9.** Denoised image with beta = 2.5

**Fig. 10.** Denoised image with beta = .5

Como podemos observar, entra las dos imagenes **Fig. 9 y Fig. 10** se puede observar que la ultima tiene menos ruido, pero se puede observar un efecto **blur** mas acentuado. Esto esta relacionado con las betas:
$\nabla E(u) = -2 * div(C * \nabla u) + 2 * \beta(u - f)$
Como podemos observar en la equacion, un escalar beta mayor castiga los cambios respecto la imagen original, provocando que la imagen con **beta 2.5** (Fig. 9) posea menos canvios en la imagen devido a esta restricción. En contrapartida la imagen

con la **beta 0.5** (Fig.10) al no tener esta restriccion produce una imagen con un efecto de denoise mas acentuado, aunque en contrapartida se prduzca un efecto de **blur** en la imagen.

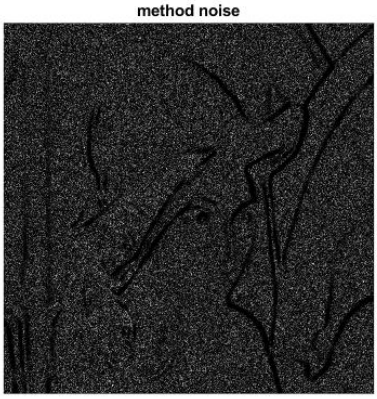



**Fig. 11.** (imagen denoise - noised image) error(beta 2.5)

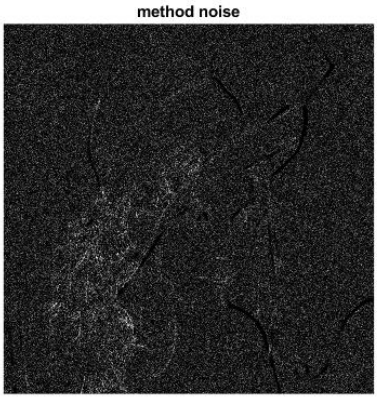



**Fig. 12.** (imagen denoise - noised image) error(beta 0.5)

Como podemos observar entre la **Fig.11 y Fig.12** la Fig.11 con una beta mayor la diferencia entre la imagen con ruido y denoised es 0 en las zonas de alta frequencia(edges), mientras que en las zonas de bajas frequencias se ha producido un efecto de denoising.
En contrapartida, la imagen con una beta menor(Fig.12) se ha producido un efecto de denoise mas acentuado que en la imagen con una beta mayor, pero no ha hecho una distincion tan clara con los edges al momento del denoising, provocando el efecto **blur** tan caracteristico.

### D. Efecto de la matriz c

**Fig. 13.** matriz c = high pass filter(**compute edge map.**)

**Fig. 14.** matriz c = ones(sz)

**Fig. 15.** Imagen generada con la matriz FIG.14

**Fig. 16.** Imagen generada con la matriz FIG.13

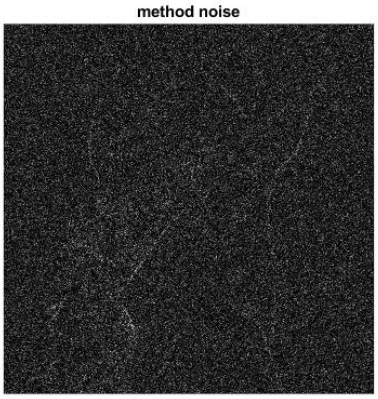



**Fig. 17.** method noise generada con la matriz FIG.14

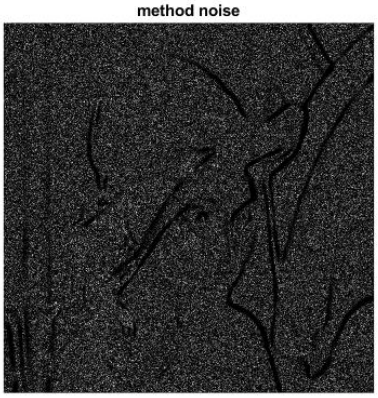



**Fig. 18.** method noise generada con la matriz FIG.13

Aunque las dos imagenes son parecidas as simple vista se observa que la imagen generada con la matriz c de la FIG.14 contiene un blur mas pronunciado que la generada con la matriz de la FIG.13.
Esto se puede apreciar de manera mas clara con la imagen method noise donde se observa como en la imagen FIG.18 generada con la matriz c de Fig.13 los edges se mantienen inalterables, resultando en una imagen con mas detalle.