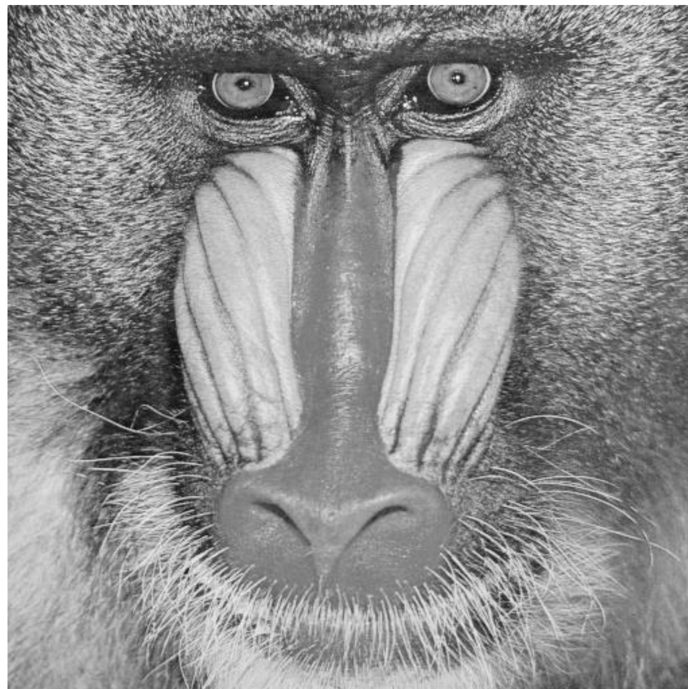
UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

Algoritmi care lucrează cu tablouri bidimensionale
Partea a II-a
12.12.2020

Matrici bidimensionale: Blur Gaussian pe imagini alb-negru

În cele ce urmează vom încerca să aplicăm un efect de blur Gaussian asupra unei imagini alb-negru. Dorim să implementăm totul de la zero, în C/C++ pentru a ilustra utilitatea cunoștințelor despre tablourile bidimensionale într-un context informatic real: prelucrarea de imagini.
În primă fază vom încerca să stabilim clar care sunt datele de intrare și particularitățile lor:
O imagine alb-negru, entitatea care urmează să fie prelucrată, nu este altceva decât o matrice bidimensională de pixeli care are un anumit număr de linii și de coloane și care înmagazinează informația referitoare la luminozitatea pixelilor din imagine sub forma de valori numerice naturale pozitive din intervalul închis 0 … 255. Pentru simplitate am ales cel mai simplu format de reprezentare al imaginilor alb-negru numit Portable Gray Map. Specificatiile acestui format pot fi studiate aici:
http://netpbm.sourceforge.net/doc/pgm.html
Vom folosi principiul singurei responsabilități, fiecare funcție având o singură responsabilitate. Astfel, este natural ca prima funcție pe care o vom implementa este cea de citire a unei matrice bidimensionale de valori reprezentate pe octeți dintr-un fisier .pgm. Am ales pentru testare următoarea imagine:



Un fișier nu este altceva decât o succesiune finită de octeți de informație stocată în memoria persistentă a calculatorului. Dacă deschidem fișierul pgm cu un editor de texte vom putea studia formatul datelor din interiorul lui. Se observă pe prima linia apare un numar magic, in cazul

formatului pgm acesta fiind P2. Pe linia următoare va fi prezent un comentariu textual legat de imagine. Linia următoare conține, separate printr-un singur spațiu, numărul de linii și de coloane ale imaginii, urmate pe linia următoare de valoarea maximă a plajei de valori folosite în reprezentarea nuanțelor de gri din imagine. În cazul de față, este valoarea standard, 255. Cunoscând numărul de linii și de coloane putem parcurge apoi rând pe rând octeții care desemnează informația referitoare la pixelii imaginii.

```
P2
# baboon.pgma created by PGMA_IO::PGMA_WRITE.
512  512
255
160  60  53  97  151  99  67  36  77  84  100  160
169  118  82  161  201  133  87  102  80  98  113  73
78  134  133  77  62  160  191  73  75  151  154  97
85  65  188  97  38  77  177  174  67  120  102  138
158  117  84  128  127  123  179  201  162  108  185  171
95  180  213  136  187  147  155  162  202  174  117  171
90  70  123  128  139  123  69  136  216  172  73  75
51  42  102  95  124  100  205  225  124  78  37  57
48  134  129  104  96  133  150  108  119  165  107  167
181  153  166  110  123  179  161  177  156  135  125  73
125  177  172  143  188  191  131  220  165  191  147  215
164  147  199  187  145  59  79  116  204  102  154  198
179  119  135  180  158  182  72  64  94  119  102  60
69  115  78  54  114  89  88  150  181  80  151  105
77  55  88  110  147  140  136  119  125  172  150  174
199  165  186  104  77  105  75  160  179  66  73  70
150  192  161  72  85  158  188  187  210  198  177  146
166  176  186  178  184  151  104  128  189  168  89  99
73  77  110  164  179  192  201  208  148  156  190  116
83  110  65  57  179  171  153  168  157  176  200  181
89  126  126  180  77  123  168  87  78  181  171  188
165  78  90  124  171  63  64  116  96  80  150  139
153  85  116  167  199  153  79  94  98  157  197  165
205  179  105  114  109  116  83  82  162  211  225  196
198  161  166  180  165  135  175  131  153  198  190  69
49  94  153  166  146  80  72  194  175  89  107  192
```

Studiind fișierul care trebuie citit cu atenție putem creiona algoritmul de citire și apoi putem implementa funcția de citire, dar nu înainte de a stabili natura datelor de ieșire. Vom folosi un tip de date structurat definit de utilizator pe care il vom numi imagine care va contine o matrice bidimensională de valori de tipul double (vom discuta această alegere în cele ce urmează) și doi întregi care reprezintă numărul de linii și de coloane de pixeli din imagine:

```
struct imagine{
        double pixeli[512][512];
        int nr_linii;
        int nr_coloane;
};
```

Acum avem toate ingredientele necesare implementării funcției de citire a matricei din fișier. Funcția va primi ca parametru de intrare un șir de caractere care va desemna calea din sistemul

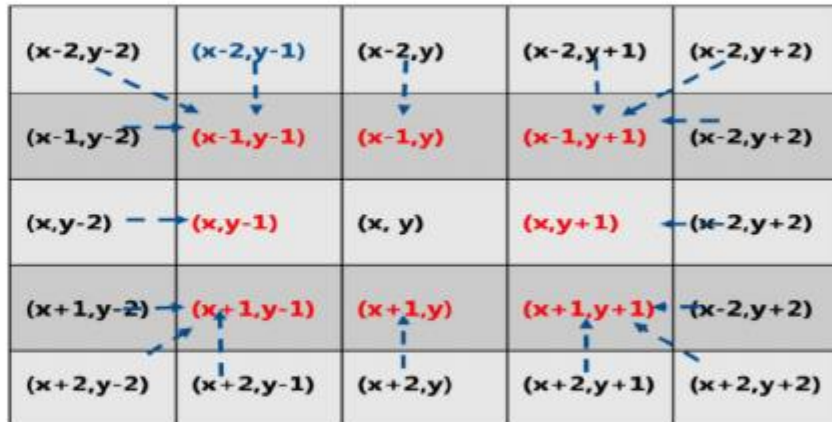
de fișiere către fișierul care trebuie citit iar ca date de ieșire o structură de tipul definit anterior care va stoca în memoria RAM toată informația despre pixelii gri din imagine. Se observă că funcția implementează exact ideea sugerată de formatul fișierului pgm: se verifică numărul magic, se tratează comentariul, se citesc pe rând numărul de linii și numărul de coloane și apoi, cu ajutorul bibliotecii stringstream se citesc pe rând, în cadrul a două cicluri for imbricate, linie cu linie și coloană cu coloană în cadrul fiecărei linii, valorile numerice din fișier care definesc nuanța de gri a pixelilor imaginii. Toate informațiile citite sunt stocate direct în atributele img.nr_linii și img.nr_coloane, respectiv img.pixeli[row][col] din entitatea de tip imagine pasată prin referință ca parametru de ieșire din funcție, img.

```
void citeste_imagine(const string& nume_imagine,imagine& img){
  int row = 0, col = 0;
  img.nr_linii = 0;
   img.nr_coloane = 0;
  ifstream infile(nume_imagine);
  stringstream ss;
  string inputLine = "";

  // verifica versiunea
  getline(infile,inputLine);
  if(inputLine.compare("P2") != 0) cerr << "Version error" << endl;
  else cout << "Versiune : " << inputLine << endl;

  // prelucreaza comentariu
  getline(infile,inputLine);
  cout << "Comentariu : " << inputLine << endl;

  ss << infile.rdbuf();
  // se citesc dimensiunile imaginii
  ss >> img.nr_linii >> img.nr_coloane;
  cout << img.nr_linii << " coloane si " << img.nr_coloane << " linii" << endl;

  // pentru fiecare linie si fiecare coloana, se citeste valoarea pixelului corespunzator
  for(row = 0; row < img.nr_linii; ++row)
    for (col = 0; col < img.nr_coloane; ++col) ss >> img.pixeli[row][col];

  infile.close();
}
```

Odată citită imaginea din fișier, informația prinde viață în memoria RAM sub formă de conductori prin care trece curent electric de o anumită intensitate. Putem astfel să începem să prelucrăm această informație.

Scopul nostru, de la început, este să aplicăm un filtru Gaussian peste imaginea inițială pentru a reduce variațiile bruște de luminozitate între pixeli adiacenți. Adiacența pixelilor este dată de vecinătatea dintre ei, așa că va trebui să stabilim clar de la început ce reprezintă această vecinătate.

Algoritmi care lucrează cu tablouri bidimensionale
Partea a II-a
12.12.2020



După cum se observă în imaginea de mai sus, un pixel situat pe linia x coloana y într-o imagine poate avea o serie de cercuri concentrice de vecini. Primul cerc, de exemplu, marcat cu roșu în imagine, reprezintă pixelii cu care pixelul central intră în contact direct. În cazul multor imagini neprelucrate apar diferite aberații de altfel normale care afectează câteodată aspectul și calitatea informației din imaginea respectivă: zgomotul din imagine și efectul de sharpness. Zgomotul este produs într-o imagine de bruiajele semnificative ale intensității luminoase ce caracterizează pixeli vecini. În demersul de față vom încerca să atenuăm acest efect prin aplicarea unui filtru asupra imaginii:

(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)	(7,1)	(8,1)
(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)	(7,2)	(8,2)
(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)	(7,3)	(8,3)
(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)	(7,4)	(8,4)
(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)	(7,5)	(8,5)
(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(6,6)	(7,6)	(8,6)
(1,7)	(2,7)	(3,7)	(4,7)	(5,7)	(6,7)	(7,7)	(8,7)
(1,8)	(2,8)	(3,8)	(4,8)	(5,8)	(6,8)	(7,8)	(8,8)

Initial image

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Kernel

$$P = (3,3) \times w_1 + (4,3) \times w_2 + \cdots + (5,5) \times w_9$$

P

Filtered image

Ideea este relativă simplă, dar eficientă. Se dorește realizarea unei alte imagini de aceleași dimensiuni cu imaginea inițială dar în care pentru fiecare pixel să se înlocuiască valoarea sa

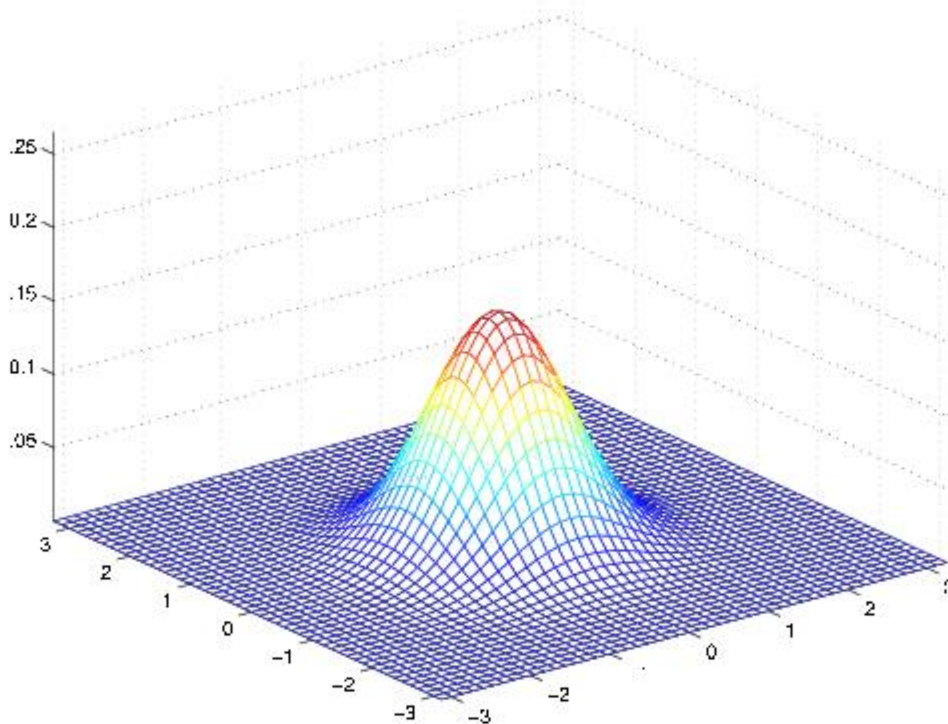
curentă cu o medie ponderată a valorilor pixelilor din vecinătatea lui, considerând că o astfel de medie va aduce valorile ieșite din comun mai aproape de trendul general de luminozitate exprimat în vecinătatea pixelului. Astfel, fiecărui pixel vecin din vecinătatea de o anumită rază a pixelului curent i se atribuie o pondere, denumită în imaginea anterioară cu wi. Deoarece i se va atribui o pondere fiecărui pixel din vecinătatea pixelului central pe o anumită rază de acțiune centrată pe acesta, toate ponderile dintr-o vecinătate de rază R vor putea fi reprezentate printr-o matrice pătratică de mărime (2*R+1) x (2*R+1) unde ponderea centrală de pe linia R+1 și coloana R+1 va fi atribuită pixelului central, iar apoi concentric, din interior spre margini se vor atribui valori ponderilor pixelilor vecini ai pixelului central.

Este destul de clar că pixelul central va avea ponderea cea mai mare și că ponderea celorlalți pixei adiacenți va fi cu atât mai mică cu cât sunt mai îndepărtați de pixelul central. Această matrice bidimensională de ponderi, numită filtru de imagine sau fereastră de convoluție, încearcă să mapeze gradul de influență pe care pixelii vecini îi au asupra pixelului central. Se poate face o paralelă între relația de vecinătate dintre pixeli și relațiile interumane dintre individ și societate: e clar că sistemul de valori al individului se calibrează în timp raportat în principal la eul său individual, apoi la valorile persoanelor imediat apropiate lui (familie, grupul de prieteni apropiați etc.), apoi la valorile dobândite în școală și de-abia apoi pe baza valorilor dobândite de la membri mai distanțați în societate față de individ.

Bazându-ne pe observațiile anterioare putem previzualiza forma unui asemenea filtru: arată ca un deal cu vârful în pixelul central a cărui pantă coboară uniform, în mod concentric, spre margini, îndepărtându-se în mod egal în toate sensurile față de acest pixel central. Am putea vizualiza forme conice sau piramidale cu această proprietate, dar aspectul trebuie să fie unul cât mai aproape de natură. Așa cum construcțiile piramidale (ziggurate, temple aztece, piramide egiptene) sunt tipic antropice, oarecum artificiale ca formă, ne vom orienta spre forma clasică de deal și anume cea de clopot. Mai precis de clopot Gaussian (https://towardsdatascience.com/a-python-tutorial-on-generating-and-plotting-a-3d-guassian-distribution-8c6ec6c41d03), pentru că dorim să modelăm o vecinătate cu influență normală asupra pixelului central. Vizual, un clopot Gaussian apare ca o suprafață neliniară într-un reper cartezian tridimensional ca în poza următoare:

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

Algoritmi care lucrează cu tablouri bidimensionale
Partea a II-a
12.12.2020



Arată foarte aproape de forma naturală de deal pe care o regăsim în natură și modelează influența naturală a pixelilor vecini asupra pixelului central. Înălțimea vârfului de deal raportată la celelalte valori și gradul de înclinare a pantei dealului pot fi parametrizate cu ajutorul variabilelor μ respectiv σ din expresia matematică a funcției care definește această suprafață:

$$G_{2\,D}(x,\; y;\, \sigma) = \frac{1}{2\,\pi\sigma^2}\, e^{-\frac{x^2+y^2}{2\,\sigma^2}}$$

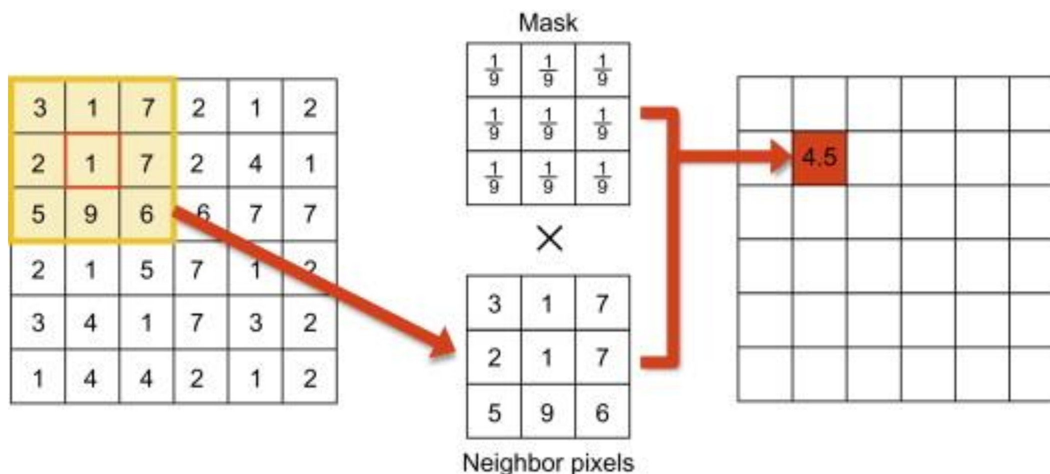
Pe baza acestor observații matematice, putem să implementăm funcția care primește doi parametri de intrare l = raza vecinătății din jurul pixelului central și σ(sigma) = deviația standard a valorilor relativ la medie și care populează o matrice de 2*l+1 x 2*l+1, numită în implementare filtru, transmisă prin referință ca parametru de ieșire, cu valorile funcției gaussiene centrate raportat la elementul de pe linia l+1 și coloana l+1, normalizate în așa fel încât suma ponderilor să fie 1. Funcția calculează valoarea conform formulei anterioare pentru fiecare pixel în primele două for-uri imbricate, adunând totodată valorile calculate. În următoarea pereche de for-uri imbricate se normalizează valorile filtrului gaussian fiind împărțite la suma tuturor valorilor din matrice. Astfel, suma ponderilor generate va fi 1.

Algoritmi care lucrează cu tablouri bidimensionale
Partea a II-a
12.12.2020

```
void genereaza_gaussian(imagine& filtru,int l,double sigma){
        filtru.nr_linii = l;
        filtru.nr_coloane = l;
        double s = 2.0*sigma*sigma;
        double sum = 0.0;
        for (int i=-l/2;i<=l/2;i++){
                for (int j=-l/2;j<=l/2;j++){
                        filtru.pixeli[i+l/2][j+l/2] = exp(-((i*i+j*j)/s))/(M_PI*s);
                        sum += filtru.pixeli[i+l/2][j+l/2];
                }
        }
        for (int i=-l/2;i<=l/2;i++){
                for (int j=-l/2;j<=l/2;j++){
                        filtru.pixeli[i+l/2][j+l/2] /= sum;
                }
        }
}
```

Având astfel filtrul pe care dorim să-l aplicăm pe imaginea inițială, vom realiza acest lucru printr-un procedeu numit convoluție: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Din nou, ideea este simplă. Pentru fiecare pixel din imagine de pe fiecare rând și de pe fiecare coloană, se calculează o sumă ponderată între valoarea pixelilor din vecinătatea pixelului central și ponderea corespunzătoare pixelului definită în matricea bidimensională a filtrului, în cazul nostru Gaussian.



De aceea funcția pentru convoluție va primi ca parametri de intrare doua entități de tip imagine și va construi o altă entitate de tip imagine ce va constitui parametrul de ieșire a funcției. Folosind

două for-uri imbricate pentru parcurgerea fiecărui element al matricii de pixeli numita img, iar în interiorul celor două for-uri alte două for-uri imbricate pentru a parcurge vecinătatea fiecărui pixel și pentru a calcula suma ponderată între pixelii din vecinătate și respectiv ponderea lor corespunzătoare din fereastra de convoluție Gaussian reprezentată de parametrul de intrare filtru și a o stoca în matricea rezultat numită rez.

Astfel, vom obține o imagine similară dar în care vor fi atenuate discrepanțele majore de luminozitate între pixeli adiacenți. Tot ce mai rămâne de făcut este să se salveze imaginea rezultată în urma procesului de convoluție într-un alt fișier pgm.

```
void convolutie(const imagine& img, const imagine& filtru, imagine& rez){
        int k = filtru.nr_linii;
        rez.nr_linii = img.nr_linii;
        rez.nr_coloane = img.nr_coloane;
        cout<<filtru.nr_linii<<" "<<filtru.nr_coloane<<endl;
        for (int i=0;i<img.nr_linii;i++){
                for(int j=0;j<img.nr_coloane;j++){
                        rez.pixeli[i][j] = 0;
                        for (int k = 0;k<filtru.nr_linii;k++){
                                for(int l = 0;l<filtru.nr_coloane;l++){
                                        if (i-k >= 0 && i-k <img.nr_linii && j-l>=0 && j-l<img.nr_coloane){

                                                rez.pixeli[i][j] += img.pixeli[i-k][j-l]*filtru.pixeli[k][l];
                                                //rez.pixeli[i][j] = img.pixeli[i][j];
                                        }
                                }
                        }
                }
        }
}
```

O problemă care trebuie tratată este faptul că pentru efectuarea corectă a calculelor, valorile numerice stocate în structura de imagine sunt de tipul double dar pentru a fi salvate în formatul pgm aceste valori trebuie retransformate în valori naturale din intervalul 0 .. 255. Putem presupune ca valorile rezultate în urma însumării ponderate a unor valori din intervalul 0 .. 255 râmâne în intervalul 0 .. 255 deoarece ponderilor folosite de noi au fost normalizate pentru ca suma lor să fie 1. Tot ce mai rămâne de făcut pentru funcția de salvare în fișier este să aproximeze valorile reale din intervalul 0 .. 255 obținute în urma aplicării filtrului de convoluție asupra imaginii la valorile întregi apropiate. Funcția a fost realizată în stilul C.

Algoritmi care lucrează cu tablouri bidimensionale
Partea a II-a
12.12.2020

```
void salvare_imagine(const string& nume_fisier,const imagine& img)
{
  FILE* pgmimg;
    pgmimg = fopen(nume_fisier.c_str(), "wb");
    double temp;
    // Writing Magic Number to the File
    fprintf(pgmimg, "P2\n");
    int width = img.nr_coloane;
    int height = img.nr_linii;
    // Writing Width and Height
    fprintf(pgmimg, "%d %d\n", width, height);

    // Writing the maximum gray value
    fprintf(pgmimg, "255\n");
    int count = 0;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            temp = img.pixeli[i][j];
            // Writing the gray values in the 2D array to the file
            fprintf(pgmimg, "%d ", int(temp));
        }
        fprintf(pgmimg, "\n");
    }
    fclose(pgmimg);
}
```

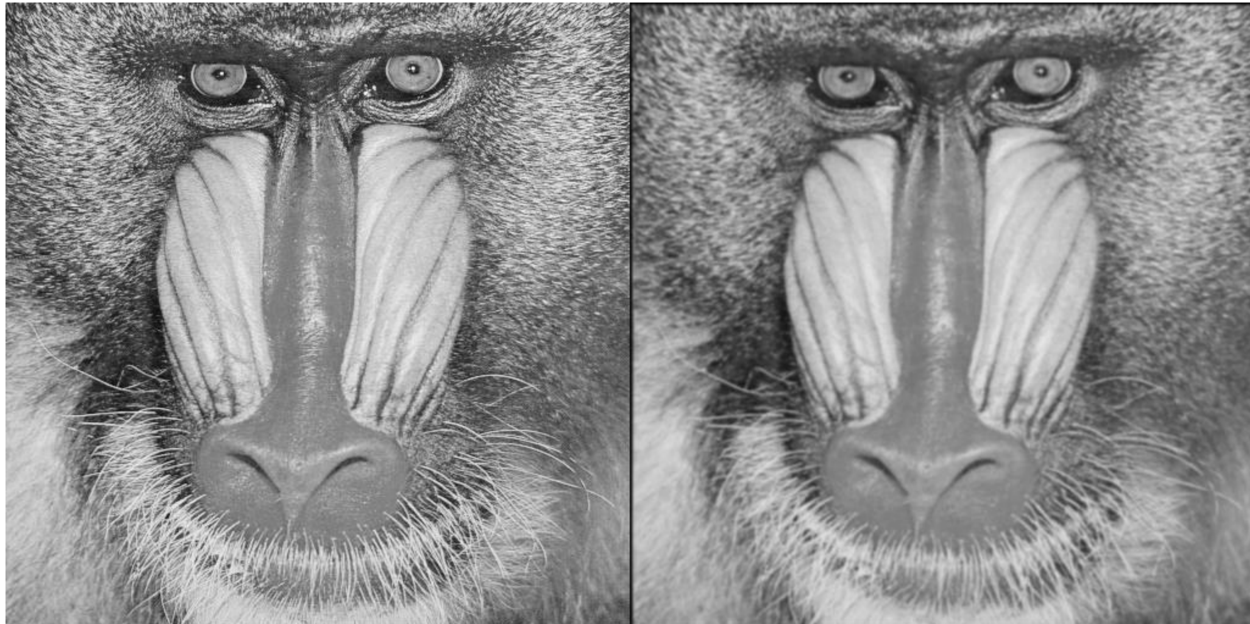
În final, toate funcțiile prezentate anterior sunt aplicate asupra imaginii de pe disc în cadrul funcției principale:

```
int main(){
imagine img;
citeste_imagine("baboon.pgm",img);
//afiseaza_imagine(img);
imagine filtru;
genereaza_gaussian(filtru,5,1.0);
afiseaza_imagine(filtru);
imagine rezultat;
convolutie(img,filtru,rezultat);
//afiseaza_imagine(rezultat);
salvare_imagine("baboon_blur.pgm",rezultat);
return 0;
}
```

Rezultatul se poate observa comparând imaginea inițială cu cea obținută în urma aplicării filtrului Gaussian de diametru 5.

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

Algoritmi care lucrează cu tablouri bidimensionale
Partea a II-a
12.12.2020



Algoritmul de convoluție stă la baza rețelelor neuronale convoluționale din ce în ce mai populare la ora actuală pentru ratele lor ridicate de detecție a diferitelor tipuri de elemente în imagini: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

Matrici bidimensionale: Rezolvarea de sisteme de ecuații liniare

După ce am observat aplicabilitatea reală a cunoștințelor despre matrici bidimensionale în informatică la nivel practic în domeniul prelucrării de imagini, vom ilustra în cele ce urmează impactul acestor cunoștințe și în domeniul matematicii.
În acest sens vom discuta algoritmul obținerii inversului unei matrice pătratice. Înainte de toate trebuie însă să definim un tip de date care să reprezinte o astfel de matrice. Îl vom numi matrix.

```
struct matrix{
        double elems[100][100];
        int grad;
};
```

În cadrul rezolvării problemei inversării unei matrici întâlnim o altă subproblemă: calcularea determinantului unei matrice pătratice de mărime variabilă. Vom folosi abordarea lui Gauss de reducere a problemei la rezolvarea recurentă a găsirii determinantului pentru matrici de grad mai mic cu 1 decât matricea inițială numite minori caracteristici, obținute prin eliminarea din matricea

inițială a tuturor valorilor aflate pe o anumită linie și o anumită coloană. Obținerea unui minor este esențială pentru calcularea determinantului matricei așa că ea va fi prima implementată:

```
void constructie_minor(const matrix& m,int l, int c, matrix& minor){
        int k=0;
        for (int i=0;i<m.grad;i++){
                for (int j=0;j<m.grad;j++){
                        if (i != l && j != c){
                                minor.elems[k/minor.grad][k%minor.grad] = m.elems[i][j];
                                k++;
                        }
                }
        }
}
```

Folosim doi indici pentru a naviga linie cu linie și coloană cu coloană în matrice, evitând linia l și coloana c, iar elementele obținute astfel vor fi așezate în ordinea în care sunt parcurse în matricea minor, de grad imediat inferior matricii de intrare m cu ajutorul contorului k și a observației că elementul k se află într-o matrice pătratică de un anumit grad pe linia k/grad și pe coloana k%grad din cadrul acelei linii.

```
double determinant(const matrix& m){

        if (m.grad == 1)
                return m.elems[0][0];
        double suma = 0;
        int factor =1;
        matrix minor;
        minor.grad = m.grad-1;
        for (int i = 0;i<m.grad;i++)
        {
                constructie_minor(m,0,i,minor);
                print_matrix(minor);
                suma += factor*m.elems[0][i]*determinant(minor);
                factor *= -1;
        }
        return suma;
}
```

Folosind metoda lui Gauss se poate calcula determinantul unei matrici pătratice de orice grad în mod recursiv: dacă matricea are grad 1, determinantul va fi unicul element din acea matrice, altfel, determinantul se va calcula ca suma dintre produsele obținute înmulțind elementele de pe o linie cu un factor care alternează între 1 și -1 și cu determinantul minorului caracteristic pentru linia și coloana acelui element calculat cu funcția definită anterior. Pentru economisirea spațiului, în implementarea propusă, minorii vor fi suprascriși de către algoritmul construcție_minor în aceeași zonă de memorie desemnată de variabila locală funcției determinant numită minor, de tipul matrix. Odată calculat determinantul matricei, putem să ne ocupăm și de generarea matricii adjuncte. Acest proces poate fi împărțit în mai mulți pași mai simpli succesivi. Inițial calculăm matricea de

cofactori care va reține pentru fiecare termen din matrice, in parametrul de ieșire de tip matrix numit cofactori doar combinatia de factor alternativ 1 și -1 înmulțită cu determinantul minorul caracteristic corespunzător.

```
void adjugare(matrix& m){
        for (int i=0;i<m.grad;i++){
                for (int j=i+1;j<m.grad;j++){
                        swap(m.elems[i][j], m.elems[j][i]);
                }
        }
}

void multiplica_scalar(matrix& m, double scalar){
        for (int i=0;i<m.grad;i++){
                for (int j=0;j<m.grad;j++){
                        m.elems[i][j]*=scalar;
                }
        }
}
```

Pentru a obține din matricea de cofactori matricea adjunctă sunt transpuse elementele matricii relativ la diagonala ei principală prin metoda adjugare și înmulțirea finală a matricii adjuncte cu scalarul reprezentat de determinantul matricii inițiale este realizată prin implementarea funcției multiplica_scalar.

```
int inverseaza(const matrix& m, matrix& inversa){
        inversa.grad = m.grad;
        constructie_cofactori(m,inversa);
        print_matrix(inversa);
        adjugare(inversa);
        print_matrix(inversa);
        double d = determinant(m);
        if (d !=0){
                //multiplica_scalar(inversa,1/d);
                return 0;
                }
        return 1;
}
```

Astfel, calcularea inversei matricei inițiale devine o formalitate, așa cum e ilustrat în imaginea anterioară. Prima dată se construiește matricea de cofactori ajutorul funcției constructie_cofactori, rezultatul intermediar fiind stocat în variabila de ieșire. Matricea de cofactori este transpusă pentru a obține in-place matricea adjunctă matricii inițiale, iar înmulțirea acestei matrici cu scalarul obținut prin calcularea determinantului o transformă pe aceasta în inversa matricii inițiale.

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

Algoritmi care lucrează cu tablouri bidimensionale
Partea a II-a
12.12.2020

Importanța matematică concretă a calculării matricii inverse unei matrici pătratice este ilustrată de folosirea acestui calcul în rezolvarea de sisteme liniare de n ecuații cu n necunoscute.
https://www.mathsisfun.com/algebra/systems-linear-equations-matrices.html
Coeficientii unui sistem de n ecuații cu n necunoscute pot fi reprezentați sub forma unei matrice pătratice de valori reale, iar atât necunoscutele cât și termenii liberi ai ecuațiilor liniare din sistem pot fi reprezentate ca tablouri unidimensionale de date.

```
void inmulteste_linie(const matrix& m, double linie[],double rez[]){
        int s;
        for (int i=0;i<m.grad;i++){
                s = 0;
                for (int j=0;j<m.grad;j++){
                        s += m.elems[j][i]*linie[j];
                }
                rez[i] = s;
        }
}
```

De aceea se implementează funcția inmulteste_linie care inmulțește o matrice pătratică cu o matrice unidimensională numită linie pentru a obține o altă matrice unidimensională numită rez.
Considerând X = matricea unidimensională a necunoscutelor, A = matricea pătratică a coeficienților reali ai celor n ecuații liniare și B = matricea unidimensională a termenilor liberi, putem scrie întreg sistemul sub forma:

$$A*X = B$$

Înmulțind la stânga cu inversa lui A, să o numim IA, obținem:

$$(IA*A)*X = IA*B$$

Cum inmultind o matrice cu inversa iei se obține elementul neutru față de înmulțire a matricilor putem ajunge la forma simplificată

$$X=IA*B$$

În alte cuvinte, tot ce trebuie să facem pentru a rezolva sistemul de ecuații, dacă acest lucru este posibil e să calculăm inversa matricii A cu algoritmul expus anterior și apoi să folosim funcția inmulteste_linie pentru a obține vectorul de necunoscute ale sistemului pe baza inmulțirii dintre matricea patratică inversă lui A și vectorul unidimensional de termeni liberi.

Algoritmi care lucrează cu tablouri bidimensionale
Partea a II-a
12.12.2020

```
int main(){
        matrix m;
        m.grad = 3;
        m.elems[0][0]=1;
        m.elems[0][1]=0;
        m.elems[0][2]=2;
        m.elems[1][0]=1;
        m.elems[1][1]=2;
        m.elems[1][2]=5;
        m.elems[2][0]=1;
        m.elems[2][1]=5;
        m.elems[2][2]=-1;
        print_matrix(m);
        double d=determinant(m);
        cout<<"determinant:"<<determinant(m)<<"gigi"<<endl;
        matrix inversa;
        inverseaza(m,inversa);
        print_matrix(inversa);
        double linie[3] = {6,-4,27};
        double rez[3];
        inmulteste_linie(inversa,linie,rez);
        cout<<">>";
        for (int i = 0;i<3;i++)
                cout<<rez[i]/d<<" ";
        return 0;
}
```