Доля П.Г.
Харківський національний університет імені В. Н. Каразіна
факультет математики і інформатики
кафедра теоретичної і прикладної інформатики
2018 р.

Математичні методи обробки цифрових зображень. Використання Octave.

Зміст

1. Вступ.

1.1. Головні задачі обробки зображень.

Монохромне зображення можна визначити як двовимірну функцію $f(x,y)$, де x і y - координати на площині зображення, а амплітуда `f` представляє інтенсивність або яскравість зображення в точці з цими координатами. Для значення `f` також використовують словосполучення рівень сірого. Кольорові зображення формуються комбінацією декількох монохромних зображень. Наприклад, в колірній системі RGB зображення складається з трьох окремих монохромних компонент (червоної, зеленої і синьої) і може бути представлено вектором з трьох інтенсивностей $\left(f_r(x,y), f_g(x,y), f_b(x,y)\right)$. З цієї причини методи і способи, розроблені для монохромних зображень, можуть бути поширені на кольорові зображення шляхом послідовної обробки трьох монохромних компонент.

Первинне зображення має неперервні x та y координати, а також неперервну амплітуду `f`. Переведення його в цифрову форму вимагає перетворення координат і амплітуди до дискретних значень. Зображення координат скінченною кількістю значень називається дискретизацією, а зображення амплітуди значеннями зі скінченного набору відліків називається квантуванням. Якщо координати x і y, а також амплітуда `f` вибираються з скінченних наборів величин, то зображення називається цифровим. З

математичної точки зору результатом дискретизації монохромного зображення є матриця чисел.

Цифрова обробка зображень має широкі сфери застосування, включаючи зорові образи. Однак на відміну від людей, які здатні сприймати лише електромагнітне світлове випромінювання видимого діапазону, комп'ютерна обробка зображень покриває практично весь спектр електромагнітних і інших хвиль. Наприклад, це можуть бути ультразвукові зображення або зображення, отримані в електронній мікроскопії.

Не існує загальновизнаної межі, яка відокремлює обробку зображень від інших суміжних дисциплін, таких як аналіз зображень або машинний зір. Іноді таке розмежування робиться за принципом, що обробка зображень характеризується присутністю зображень на вході і виході системи. У всьому різноманітті задач від обробки зображень до машинного зору немає чітких меж, проте тут можна виділити комп'ютеризовані процеси низького, середнього і високого рівня. Процеси низького рівня включають лише примітивні операції над зображеннями типу зменшення шуму або поліпшення контрастності. Вони характеризуються тим, що на вході і виході присутні зображення. Процеси середнього рівня пов'язані з такими завданнями, як виділення (сегментація) і класифікація (розпізнавання) частин зображення. У середньорівневих процесах на вхід подається зображення, а на вихід надходять атрибути і ознаки, здобуті з цих зображень, наприклад, межі, контури або інші відмітні ознаки об'єктів, які також можуть бути зображеннями. Процеси високого рівня займаються «осмисленням» множини розпізнаних об'єктів.

Якщо ототожнити цифрове монохромне зображення з матрицею M, то обробка зображення буде полягати у перетворенні цієї матриці в матрицю $\tilde{M}$ того ж розміру (або приблизно такого ж розміру). Тобто перетворення $\tilde{M} = f(M)$ умовно відноситься до процесів низького рівня. Під процесами середнього рівня ми повинні розуміти перетворення $\{M_i\}_{i=1}^{n} = f(M)$ матриці M в масив матриць M_i меншого розміру, кожна з яких також є зображенням. Якщо у відображенні $\{M_i\}_{i=1}^{n} = f(M)$ навчитися кожну матрицю відносити до певного класу, якому призначити ознаку, то це можна розглядати як аналіз зображень. Таким чином, відображення $\{P_i\}_{i=1}^{n} = f(M)$, де P_i - деяка ознака, відноситься до аналізу зображень, тобто до процесів обробки зображень високого рівня, що і називається розпізнаванням образів.

Для розуміння методів обробки зображень важливо розв'язувати практичні завдання, а для цього треба мати комп'ютерний інструмент, здатний ілюструвати застосування тих чи інших методів. У нашому курсі ми будемо використовувати пакет Octave. Ви можете використовувати MatLab, бо Octave був написаний з урахуванням сумісності з MatLab і реалізує більшість його можливостей. Різниці в мові програмування майже немає. Навіть пакети розширення спроєктовані таким чином, щоб схожі математичні перетворення виконувалися однойменними функціями з мінімальною різницею між їх аргументами.

MatLab і Octave є середовищами для виконання технічних і наукових обчислень. Їх базовою конструкцією є масив елементів (матриця), який не вимагає задання розмірності. Це дозволяє легко формулювати умови і будувати розв'язки багатьох обчислювальних задач, що потребують матричного зображення. Обидві системи мають розширення у вигляді наборів спеціалізованих програм, які по-англійськи називаються toolbox (набір інструментів або пакет розширення). Більшість функцій Octave, призначених для роботи з зображеннями, зосереджено в пакеті `'image'`, який з деякими обмеженнями є двійником пакету `'Image Processing Toolbox'` MatLab. Ці пакети розширюють можливості стандартного середовища для розв'язання задач цифрової обробки зображень.

1.2. Графічні функції Octave.

Створимо матрицю `5 x 5`.

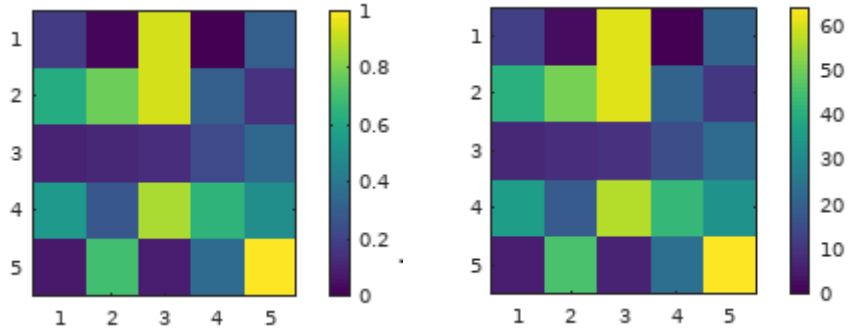
```
>>A=[12 2 60 0 20; 40 50 60 20 10; 7 8 9 15 22; 35 18 56 42 32; 5 45 6 23 64]
A =
   12    2   60    0   20
   40   50   60   20   10
    7    8    9   15   22
   35   18   56   42   32
    5   45    6   23   64
```

Для графічного зображення масивів в Octave призначена функція `image`.

```
>>image(A);
>>colorbar          % наступний рисунок ліворуч
```

Щоб шкала кольорів відповідала реальним значенням елементів матриці треба використати функцію `image` з опціями масштабування.

```
image(A,'CDataMapping','scaled'); colorbar  % наступний рисунок праворуч
```



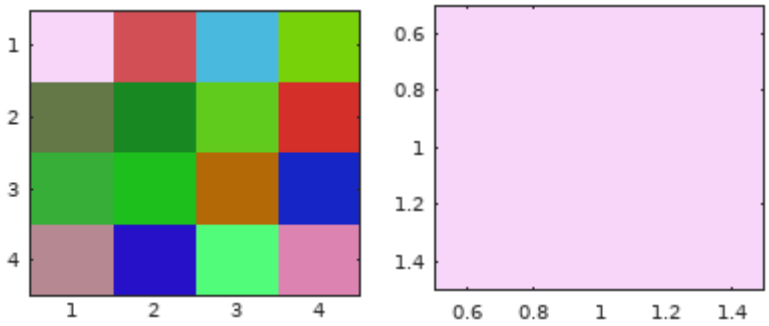
На рисунку показано частину графічного вікна Octave. Зображення представляє зону цього вікна, яка складена з `5 x 5` прямокутників, кожен з яких зафарбований кольором, номер якого вказано в елементі матриці. Кольори для номерів беруться з спеціальної таблиці, яка називається палітрою.

Масиви, які складаються з трьох шарів матриць інтерпретуються інакше. Наприклад,

```
>> C=rand(4,4,3)
C =
ans(:,:,1) =
```

```
   0.973477   0.826962   0.289170   0.473824
   0.392033   0.097783   0.382250   0.834109
   0.215221   0.110805   0.709497   0.090060
   0.716837   0.151091   0.314316   0.870308
ans(:,:,2) =
   0.842560   0.312931   0.728813   0.824929
   0.471020   0.534795   0.799199   0.186354
   0.683095   0.749376   0.414491   0.151677
   0.537297   0.067369   0.991961   0.515339
ans(:,:,3) =
   0.981093   0.335135   0.872385   0.036542
   0.276375   0.137509   0.116894   0.160076
   0.221233   0.110639   0.029313   0.777089
   0.573522   0.786834   0.475162   0.700426
```

>> image(C) % наступний рисунок ліворуч



Кожна трійка чисел, по одній з кожного шару (колірної площини), визначає RGB колір відповідного прямокутника. Наприклад трійка чисел

```
>> C(1,1,:)
ans =
ans(:,:,1) =  0.97348
ans(:,:,2) =  0.84256
ans(:,:,3) =  0.98109
```

визначає RGB колір лівого верьхнього прямокутника зображення.

>>image(C(1,1,:)) % попередній рисунок праворуч

Більшість функцій Octave, які призначені для роботи з графічними зображеннями зібрано в пакеті розширення `'image'`. Щоб його завантажити потрібно виконати команду

```
>> pkg load image
```

Розглянемо деякі функції з цього пакету. Знову створимо матрицю `5 x 5`.

```
>>A=[1 2 3 0 0; 4 5 6 0 0; 7 8 9 0 0; 0 0 0 0 0; 0 0 0 0 0]
A =
    1    2    3    0    0
    4    5    6    0    0
    7    8    9    0    0
    0    0    0    0    0
    0    0    0    0    0
```

Для її зображення використаємо функцію `imshow` з пакету `'image'`.

```
>>imshow(A);
```

Якщо матриця має тип `double`, то функція `imshow` будь яке число ≥ 1 відображає білим кольором, нуль – чорним, проміжні значення від 0 до 1 відображаються відтінками сірого. За замовчуванням числові матриці створюються типу `double`. Оскільки лівий верхній квадрат `3 x 3` матриці A містить числа більші за одиницю, тому всі відповідні прямокутники зображення мають білий колір.

Змінемо деякі значення нашої матриці.

```
>>A(1,1)=0.2;A(2,2)=0.4;A(3,3)=0.6;A(4,4)=0.8;A(4,2)=0.7;A(2,4)=0.3
>>A =
   0.20000   2.00000   3.00000   0.00000   0.00000
   4.00000   0.40000   6.00000   0.30000   0.00000
   7.00000   8.00000   0.60000   0.00000   0.00000
   0.00000   0.70000   0.00000   0.80000   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000
>>imshow(A);
```

Функція `mat2gray(A)` переводить елементи матриці `A` до діапазону `[0 1]`. Причому найбільшому елементу матриці буде відповідати одиничне значення (тобто білий колір при відображенні функцією `imshow`)

```
>>B=mat2gray(A)
B =
   0.02500   0.25000   0.37500   0.00000   0.00000
   0.50000   0.05000   0.75000   0.03750   0.00000
   0.87500   1.00000   0.07500   0.00000   0.00000
   0.00000   0.08750   0.00000   0.10000   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000
>>imshow(B);
```

Функція `im2uint8(B)` перетворює матрицю-аргумент до діапазону `[0 255]`, змінюючи тип елементів на `uint8` (unsigned intger 8 бітів).

Функція `imshow` кольори пікселів при зображенні цілочислових матриць тлумачить схожим чином з функцією `image`, тобто як номери кольорів в матриці палітри.

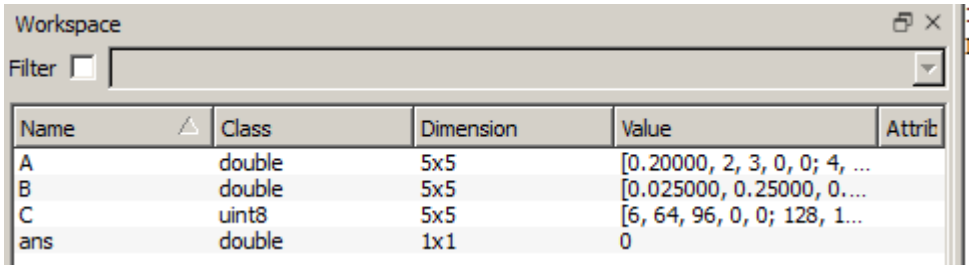
```
>>C=im2uint8(B)
```

```
C =
     6    64    96     0     0
   128    13   191    10     0
   223   255    19     0     0
     0    22     0    26     0
     0     0     0     0     0
>>imshow(C);
```

Функція `colormap` змінює поточну палітру, одночасно змінюючи кольори в активному графічному вікні.

```
>>colormap(gray);      % наступне зображення ліворуч
>>colormap(summer);    % наступне зображення в середині
>>colormap(winter);    % наступне зображення праворуч
```

Подивіться у вікні «Workspace» на тип/class матриць: `A – double`, `B – double`, `C – uint8`.



В системі є також функція `im2uint16(A)`, яка переводить елементи матриці A в діапазон `[0 65535]`.

Для завантаження зображення в робочий простір Octave використовується (загальна, не з пакету `'image'`) функція `imread` з наступним синтаксисом:

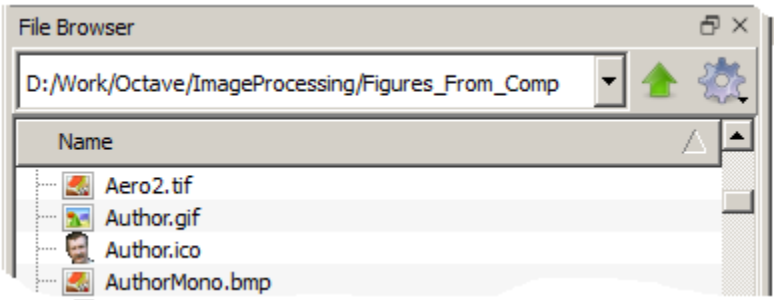
```
>>f=imread('filename')
```

Тут `'filename'` - це рядок символів, який утворює повне ім'я завантаженного файла. Наприклад, командний рядок

```
>>f=imread('D:/Work/Octave/Pictures/football.jpg');
```

присвоює матричній змінній `f` зображення з файлу «`football.jpg`». Слеши в шляхах доступу повинні бути зорієнтовані в зворотний бік. Символ `'`(апостроф) використовується в якості обмежувача символьного рядка. Крапка з комою в кінці командного рядка вказує системі не відображати виведення результату виконання операції в командне вікно. Якщо крапка з комою буде відсутня, то Octave надрукує матрицю `f`. Символ запрошення `>>` позначає початок командного рядка у вікні команд Octave. Цей символ в прикладах нашого конспекту ми, як правило, писати не будемо.

Якщо в ім'я файлу не включена інформація про каталог, то файл шукається в поточній папці, яка вибрана у вікні `File Browser`.



Якщо файла в поточній папці нема, то його пошук виконується в усіх папках, які вказані в змінній середовища IMAGE_PATH. Її можна прочитати і змінити наступним чином.

```
s=IMAGE_PATH()
s = .;C:\Octave\OCTAVE~1.1\share\octave\4.2.1\imagelib
IMAGE_PATH(strjoin({s, "D:/Work/Octave/Pictures"}, pathsep));
```

В даному посібнику ми припускаємо, що всі графічні файли зібрані в каталогах, на які у нас з самого початку встановлені шляхи пошуку. Тоді читання файла в змінну `f` можна виконувати без зазначення повного шляху.

```
f=imread('football.jpg');
```

Щоб побачити зображення потрібно виконати команду

```
imshow(f);
```

Функція `size(f)` (загальносистемна, не з пакету `'image'`) повертає розмір зображення, тобто кількість рядків, стовпців і колірних площин масиву, що представляє зображення:

```
size(f)
ans =
   256   320     3
```

Ця функція особливо корисна при зберіганні розміру зображення.

```
[M,N,K] = size(f);
```

При такому запису змінній `M` буде присвоєно кількість рядків зображення, змінній `N` - кількість стовпців, а `K` - кількість колірних площин.

Функція `whos` повідомляє додаткову інформацію про масив. Наприклад, команда

```
whos f
```

видає наступний результат:

```
Variables in the current scope:
   Attr Name      Size                Bytes  Class
   ==== ====      ====                =====  =====
        f      256x320x3              245760  uint8
Total is 245760 elements using 245760 bytes
```

Загальносистемна функція `numel(f)` повертає кількість елементів масиву `f`. Для монохромного зображення (тобто, якщо `f` матриця) це співпадає з кількістю його пікселів.

```
numel(f)
ans =  245760
```

Як ми казали, завантажене зображення можна вивести на дисплей комп'ютера за допомогою функції `imshow`, яка має наступний синтаксис:

```
imshow(f);
```

де `f` - це дійсна матриця. Команда

```
imshow(f,[low high])
```

всі елементи зі значеннями, які не перевищують число `low`, зображає чорним кольором, а всі пікселі зі значеннями більшими за число `high` - білими. Всі значення розташовані між `low` і `high` показуються з проміжною яскравістю.

```
imshow(B, [0.1,0.7]);
```

Запис в командному рядку

```
imshow(f,[ ])
```

задає для змінної `low` мінімальне значення масиву `f`, а змінній `high` присвоює максимальне значення. Така форма функції `imshow` буває корисною при показі зображень, що мають вузький динамічний діапазон, або коли в матриці `f` є від'ємні значення. Наприклад,

```
D=[-1 1 2; 0 -2 1; 0 1 2]
D =
  -1   1   2
   0  -2   1
   0   1   2
imshow(D)          % left picture
imshow(D,[ ])      % right paicture
```

У цьому посібнику ми будемо вивчати методи обробки зображень. Але після цього воно повинно бути збережено в графічний файл. Це можна виконати командою

```
imwrite(A, filename, fmt)
```

яка зберігає образ A в файл з ім'ям filename в графічному форматі, визначеному рядковим аргументом fmt. Наприклад, для зображення футбольного м'яча

```
f=imread('football.jpg');    % зазвичай відбувається обробка зображення f
```

наступна команда створить файл того ж зображення, але в форматі bmp.

```
imwrite(f,'football.bmp','bmp');
```

Зверніть увагу, що файл зберігається в поточному каталозі.

Існує кілька способів інтерпретування числових значень матриць як яскравості пікселів. Якщо елементи є додатними цілими, то ці числа можна тлумачити як індекси (номери) кольорів в іншій матриці, яка зветься картою або палітрою кольорів (colormap). Карта кольорів є $m \times 3$ матрицею дійсних чисел, діапазон зміни яких від 0 до 1. Її k-й рядок визначає колір в форматі $[r(k)\ g(k)\ b(k)]$ з відповідними інтенсивностями червоного r(k), зеленого g(k) і синього b(k) кольорів. При використанні функцій відображення матриць imshow(A) та image(A) кожний елемент матриці A зі значенням k визначає колір пікселя, беручи його з k-го рядка матриці палітри. Отже функції imshow(A) та image(A) розглядають елементи цілочислової матриці A як індекси в карті (палітрі, матриці) кольорів і відображають їх відповідним чином.

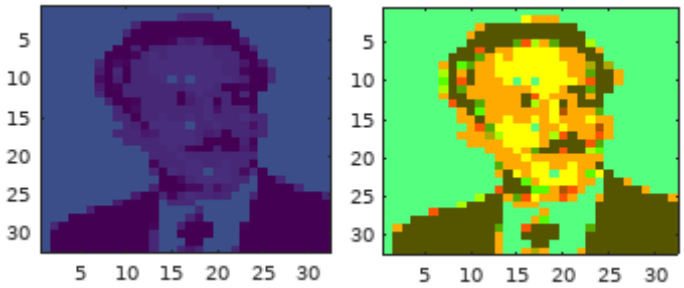
Розглянемо для прикладу малюнок, який знаходиться в графічному файлі Pogorelov32x32.bmp, розміром 32 на 32 піксела. Команда

```
A=imread('Pogorelov32x32.bmp');
```

читає і друкує матрицю A розміром 32 x 32. Значення елементів матриці A є номерами кольорів відповідних пікселів. Команда

```
image(A);
```

відображає цю матрицю в графічному вікні (наступний рисунок ліворуч), беручі кольори з палітри за замовчуванням.



Змінимо палітру наступною командою

```
colormap(colorcube)    % попередній рисунок праворуч
```

Команда

```
colormap('default')
```

повертає палітру за замовчуванням.

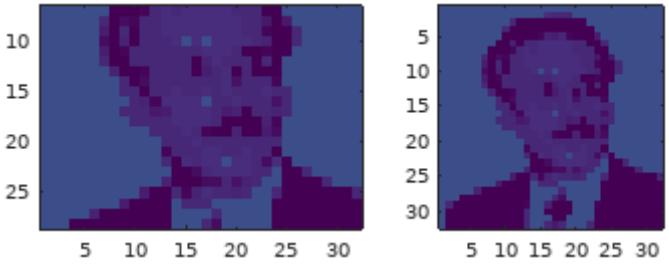
Малюнок не дуже добрий і для отримання прийнятного зображення потрібно змінити параметри його відображення. В нашому прикладі малюнок витягнутий по горизонталі. Команда

```
axis equal;
```

вирівнює горизонтальні і вертикальні розміри пікселів в графічному вікні. Вона встановлює коефіцієнт стиснення зображення однаковим по всіх осях (наступний малюнок ліворуч). Кращий результат дає команда

```
axis image;
```

бо охоплюючий зображення прямокутник в графічному вікні деформується під розмір зображення (наступний рисунок праворуч).



Кольори точок беруться з поточної карти кольорів. Як ми казали карта кольорів (палітра) - це матриця, яка містить 3 стовпці і кілька (може бути багато) рядків. Наприклад, команда

```
gray(5)
```

створює наступну матрицю, яка може бути використана як палітра кольорів з п’ятьма відтінками сірого (включаючи чорний і білий кольори).

```
ans =
   0.00000   0.00000   0.00000
   0.25000   0.25000   0.25000
   0.50000   0.50000   0.50000
   0.75000   0.75000   0.75000
   1.00000   1.00000   1.00000
```

Поглянемо на матрицю A. Ось її шматок.

```
>> A(20:25,15:20)
ans =
   7   8   7   7   7   7
   7   7   8   8   7   7
   8   7   8   8  15   8
   0   2   7   8   8   8
   0   0   2   2   7   3
   1   6   8   7   7   6
```

Матриця A, яка прочитана з файлу `Pogorelov32x32.bmp`, містить числа від 0 до 15. Для того, щоб переконатися в цьому достатньо виконати команди

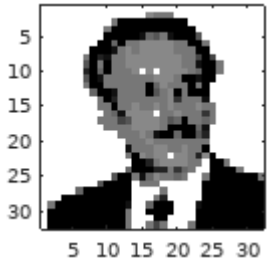
```
min(min(A))
ans = 0
max(max(A))
ans = 15
```

Функції `max` і `min` визначають найбільше і найменше значення матриці по одній з її розмірностей. Наприклад, одноразове застосування команди `max(A)` повертає вектор з 32 чисел, які є максимальними значеннями по стовпцям матриці A. Для визначення максимального значення всієї матриці потрібно ще раз використати функцію `max` до результату її першого застосування.

Оскільки матриця A містить числа від 0 до 15, то логічно створити чорно - білу палітру з 16 відтінками сірого. Команда

```
colormap(gray(16));
```

змінить палітру і перетворить малюнок в графічному вікні до наступного вигляду (рисунок ліворуч).



Зауважимо, що функція `imshow(A,[ ])` побудує те ж саме зображення (попередній рисунок праворуч), але без рамки навкруги.

Зверніть увагу на тип елементів матриці A у вікні Workspace - `uint8` (беззнакові 8 - бітні цілі змінюються в діапазоні від 0 до 255).

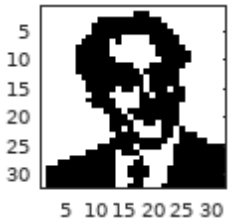
Якщо виконати команду

```
B=A/15;
```

то операція ділення цілих чисел дасть тільки 0 або 1, а тип елементів матриці B не зміниться (ціле розділити на ціле дає ціле). Це дозволяє створити чорно - білий малюнок. Для цього побудуємо палітру з двох кольорів - чорного (йому відповідає рядок 0, 0, 0) і білого (йому відповідає рядок 1, 1, 1).

```
cm=[0 0 0;1 1 1];   % матриця палітри (два кольори – чорний і білий)
image(B);
axis image; colormap(cm);
cm
```

```
cm =

     0     0     0
     1     1     1
```



Зображення з самого початку могло бути чорно-білим. Наприклад, автор створив в Paint чорно-білу картинку `32 x 32` пікселя `AuthorMono.bmp`. Завантажте цей файл і подивіться у вікні Workspace на тип матриці `f` - `logical`.

```
f=imread('AuthorMono.bmp')
```

Матриця `f` складається з логічних нулів та одиниць.

```
imshow(f);
```

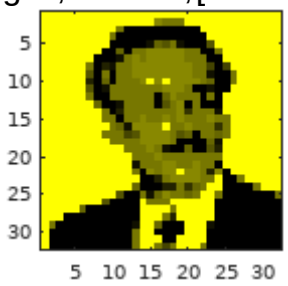
Теж саме зображення буде побудовано інструкціями

```
image(f);
axis image; colormap(cm);
```

Якщо масив A є тривимірним з трьома кольорними площинами і елементами, які змінюються в діапазоні від 0 до 1, то функції `image` та `imshow`

інтерпретують його як RGB зображення. Наприклад, склавши тривимірний масив з двох однакових колірних компонент і нульової третьої, ми побачимо монохромне жовте зображення.

```
A0=zeros(size(A));
A1=mat2gray(A);
A3=cat(3,A1,A1,A0);
image(A3);  axis image; set(gcf,'Color',[1 1 1]);
```



Щоб дізнатися формат зображення можна виконати команду

```
info = imfinfo('Pogorelov32x32.bmp')
```

Вона повертає структуру з великою кількістю інформації про графічний файл. Тут ми наводимо деякі поля цієї структури

```
info =
  scalar structure containing the fields:
    Filename = D:/Work/Octave/Pictures\Pogorelov32x32.bmp
    FileModDate = 29-Mar-2012 19:38:40
    FileSize =  630
    Format = BMP
    FormatVersion =
    Width =  32
    Height =  32
    BitDepth =  8
    ColorType = indexed
  . . . . . .
```

Зверніть увагу на те, що далі друкується матриця палітри, а також на рядок з параметром `ColorType`, в якому вказується тип графічного файлу. У нашому посібнику ми зустрінемося зі значеннями цього параметра `indexed`, `grayscale`, `binary` і `RGB`. Зараз ми говоримо про графічні файли типу `indexed`, тобто ті, які використовують кольорову палітру.

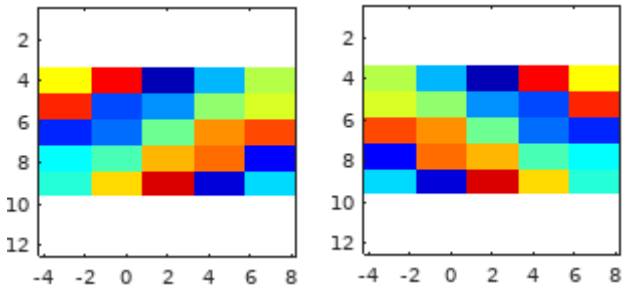
Команда `image(x,y,C)`, де `x` і `y` є двоелементними векторами `[min,max]`, які визначають діапазон зміни координат, малює такий же образ, що і команда `image(C)`, але масштабований. Це буває корисно, наприклад, в разі, коли ви бажаєте, щоб мітки осей відповідали реальним фізичним координатам графічного образу.

```
A = magic(5);
x=[-3 7]; y=[4 9];
image(x,y,A), axis equal, colormap(jet(25))  % наступний рисунок ліворуч
```

Зауваження. Функція `magic(n)` створює цілочислову «магічну» матрицю розміру `n x n`, у якої сума елементів по рядкам, стовпцям і діагоналям однакова.

Якщо діапазон задано у вигляді `[max, min]`, то зображення перевертається навколо відповідної осі.

```
x=[7 -3]; y=[4 9];
image(x,y,A), axis equal, colormap(jet(25))  %наступний рисунок праворуч
```



Слово `jet` представляє ім'я карти кольорів. Назви визначених в системі палітр можна відшукати на довідковій сторінці функції `colormap`.

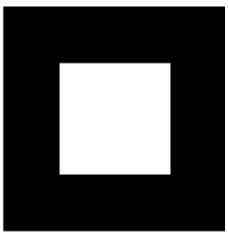
```
>> help colormap
```

При чому ім’я палітри можна використовувати з аргументом – кількістю кольорів (рядків) в матриці. Наприклад

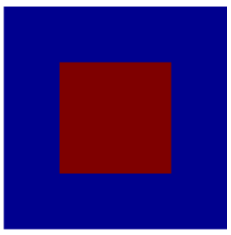
```
summer(4)
ans =
   0.00000   0.50000   0.40000
   0.33333   0.66667   0.40000
   0.66667   0.83333   0.40000
   1.00000   1.00000   0.40000
```

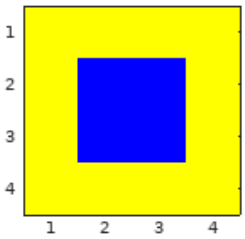
Зазвичай функція `imshow(M)` будує графічний образ напівтонової (тобто монохромної з різними градаціями яскравості) `double` матриці M. Але подальше виконання функції `colormap` може вплинути на зображення, змінивши кольори.

```
I = zeros(4,4)     % генерування нульової матриці розміром 4 x 4
I =  0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
I(2:3, 2:3) = 1    % зміна значень елементів матриці
I =  0     0     0     0
     0     1     1     0
     0     1     1     0
     0     0     0     0
imshow(I);              % наступний рисунок a
colormap('default');    % наступний рисунок b
colormap(jet);          % наступний рисунок c
```


a


b


c



d

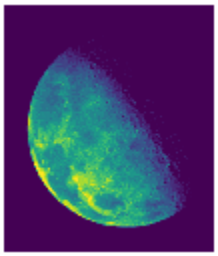
Щоб зобразити двозначну матрицю `I` за допомогою функції `image` зручно створити двокольорову палітру і перевести матрицю зображення до типу `uint8`.

```
cm=[1 1 0; 0 0 1];
image(uint8(I));
colormap(cm);        % попередній рисунок d
```

Зауваження. Якщо відображався тривимірний масив, і кольори пікселів будувалися по RGB схемі, то зміна палітри не буде впливати на зображення.

Нагадаємо, що функція `imshow` матрицю типу `uint8` також відображає як `indexed`, тобто використовує карту кольрів. Ось ще приклад використання функції `imshow`.

```
moon = imread('moon.jpg');
imshow(moon)
colormap(gray(255));          % наступний рисунок ліворуч
```

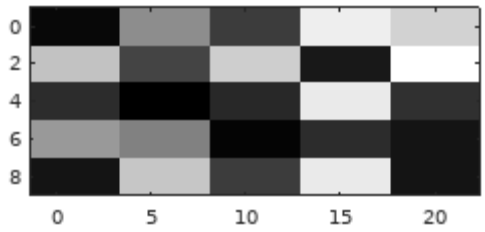
Можна відразу використати ім'я файлу

```
imshow('moon.jpg'); colormap('default')   % попередній рисунок праворуч
```

З різними варіантами використання функції `imshow` слід познайомитися по довідковій системі.

Існує ще одна функція `imagesc(x,y,A)`, яка відображає матрицю `A` на координатній сітці з діапазонами, що визначаються векторами `x` та `y`, і кожен елемент матриці відображається прямокутником.

```
A=rand(5,5)   % випадкова матриця розміром 5 x 5 з елементами від 0 до 1
A =
   0.061625   0.530196   0.241486   0.861561   0.761358
   0.704416   0.271993   0.740873   0.121917   0.922901
   0.189480   0.025966   0.177225   0.848281   0.198772
   0.560578   0.481922   0.048139   0.190799   0.104167
   0.098060   0.723835   0.248569   0.845579   0.101954
imagesc([1,20],[0,8],A); axis image
```



Команда

`[X,map] = imread(filename)`

читає графічний файл в матрицю `X`, а його карту кольорів в матрицю `map`. Наприклад,

```
[A,m] = imread('Pogorelov32x32.bmp');
image(A); colormap(m);
```

Інструкція

`[A,map,alpha] = imread(...)`

в третьому параметрі повертає маску (матрицю, елементами якої є числа 0 і 255), яка використовується для визначення прозорості точок образу. Елементи матриці `alpha`, які дорівнюють 0, вказують на те, що відповідні точки зображення бажано не фарбовувати. Цей параметр можна використати при зображенні іконки (значка).

```
[A,B,C]=imread('Author.ico');
image(A);axis image;                % наступний рисунок ліворуч
```

При створенні файла іконки одночасно з зображенням створюється матриця маски. Її нульові елементи «умовно» позначають точки фону значка, які при його малюванні не повинні відображатися в прямокутнику зображення (повинен залишатися фон екрану). Деякі функції API Windows вміють це робити, а функція `image` – ні. Однак матрицю маски ми можемо читати і використовувати.

Зробимо фон зображення білим. Спочатку створимо тривимірний масив цілих, заповнений числом 255.

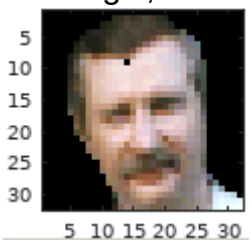
```
D = ones(size(A)) * 255;
size(D)
ans =
   32    32     3
```

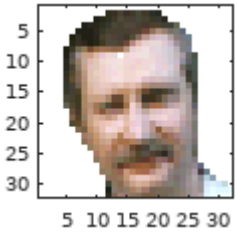
Потім перенесемо з масиву `A` в масив `D` значення тільки тих пікселів, для яких в масці `C` елементи дорівнюють `255`.

```
C3=cat(3,C,C,C);
D(C3 == 255) = A(C3 == 255);
```

Нарешті відобразимо зображення скоректованого масиву `D`.

```
image(uint8(D));axis image;       % наступний рисунок праворуч
```





Щоб пояснити роботу останніх інструкцій спочатку розглянемо простіший приклада.

```
E=uint8([1 2 3; 4 5 6; 7 8 9])      % створюємо uint8 матрицю
E =
  1  2  3
  4  5  6
  7  8  9
E0=zeros(size(E))                   % створюємо нульову матрицю
E0 =
   0   0   0
   0   0   0
   0   0   0
CL=[1 0 0; 0 1 0; 0 0 1]==1         % створюємо логічну матрицю
CL =
  1  0  0
  0  1  0
  0  0  1
E(CL)'
ans =
  1  5  9
```

Операція `им'я_матриці(логичний масив)` називається логічним індексуванням, де `логічний масив` повинен мати той же розмір, що і `матриця`. Результатом операції логічного індексування є вектор, складений з елементів вхідного масива, для яких в логічному масиві відповідні елементи дорівнюють логічній одиниці. Цей вектор може стояти як в правій частині оператора присвоювання, так і в лівій.

```
E(CL)=E0(CL)
E =
  0  2  3
  4  0  6
  7  8  0
```

Інструкція `E(CL)` вибирає (створює одновимірний вектор) з матриці E тільки ті елементи, для яких елементи логічної матриці `CL` дорівнюють одиниці. Команда `E(CL)=E0(CL)` копіює однаково розташовані в матрицях `E` і `E0` елементи з вектора `E0(CL)` до вектора `E(CL)`, одночасно замінюючи відповідні елементи матриці `E`. В результаті за допомогою логічного індексування з матриці `E0` в матрицю `E` скопіювалися тільки діагональні елементи.

Повернемося до попереднього прикладу, в якому ми змінили колір фону іконки. Ось його головні інструкції.

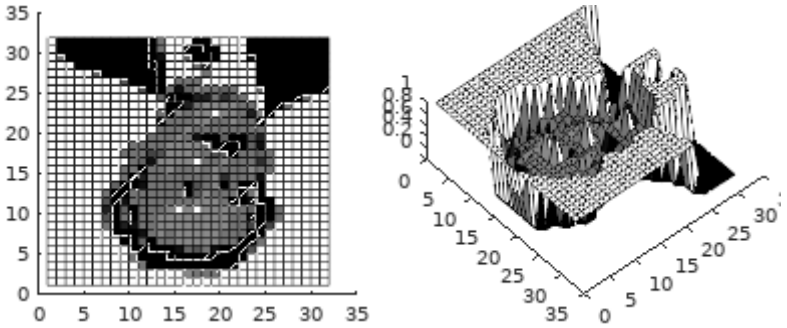
```
D = ones(size(A)) * 255;
C3=cat(3,C,C,C);
D(C3 == 255) = A(C3 == 255);
image(uint8(D));
```

Спочатку ми створили `double` масив `D` з чисел 255. Якщо подивитися у вікні Workspace, то побачимо, що матриця `C` має тип `uint8` і складається тільки з

чисел 0 та 255. Вона має такий же розмір, що і одна кольорова площина масива A. Оскільки масив D тривимірний, то нам потрібен тривимірний логічний масив. Тому спочатку з матриці `C` ми створили тривимірний масив `C3`. Тоді вираз `C3==255` являє тривимірний логічний масив, який використовується зліва і справа в операції присвоювання. В результаті з масива `A` в масив `D` копіюються тільки ті значення, для яких в масиві `C3` елементи дорівнюють 255. Елементи масива `D`, для яких відповідні елементи в масці дорівнювали 0, залишаються незмінними, тобто 255 (білими). Оскільки елементи масиву `D` змінюються від 0 до 255, то щоб функція `image` вірно відобразжала кольори, її треба перетворити до типу `uint8` (бо початковий масив A має цей тип).

Іноді бажано відобразити матрицю як поверхню. Це можна зробити за допомогою функції `surface(Z)`, де матриця `Z` містить дійсні числа, які будуть інтерпретуватися як висота поверхні над площиною XY. Якщо матриця, прочитана з графічного файлу і містить цілі числа, то може знадобитися перетворення типу.

```
A = imread('Pogorelov32x32.bmp');
C = mat2gray(A);
surface(C); colormap(gray);
```



Щоб побачити зображення, яке показано на попередньому рисунку праворуч, треба вгорі графічного вікна натиснути кнопку  Rotate і покрутити мишею зображення.

Аналогічно працюють команди `surf(Z)` і `surfc(Z)`, які створюють тривимірний графік поверхні по `z` координатам, заданим в матриці `Z`, використовуючи `x=1:n` і `y=1:m`, де `[m,n]=size(Z)`.

Корисною може бути також функція побудови каркасного зображення поверхні. Один з найбільш часто використовуваних її синтаксисів наступний

```
mesh(X,Y,Z);
```

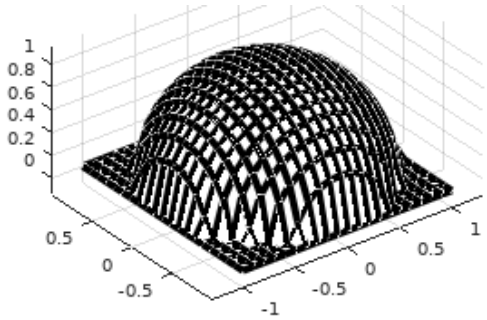
де матриці X, Y, Z повинні бути однакового розміру. Графік поверхні будується над прямокутною областю площини XY, яка розбивається сіткою розміром таким же, як у матриць. У вузлах сітки значення функції дорівнюють відповідним значенням елементів матриці Z. Матриці X і Y містять `x` і `y` координати вузлів сітки.

У наступному прикладі ми будуємо одиничну півсферу по рівнянню

$$z = \begin{cases} \sqrt{1-x^2-y^2}, & x^2+y^2 \le 1 \\ 0 \quad , & x^2+y^2 > 1 \end{cases}$$

де $-1 \le x \le 1$, $-1 \le y \le 1$.

```
[X Y]=meshgrid(-1:0.1:1);
XY=X.^2+Y.^2;
C=XY<=1;
XYZ=zeros(size(XY));
XYZ(C)=1-XY(C);
Z=sqrt(XYZ);
mesh(X,Y,Z);    % mesh(X,Y,Z,'LineWidth',4); colormap(gray(1));
axis equal
```



Тут ми використали логічне індексування так само, як описано раніше.

Перелічимо функції, які розглянуті в цьому параграфі.

`imread('файл',...)`	читання графічного файлу
`imwright(...)`	збереження графічного файлу
`whos`, `imfinfo('файл',...)`	інформація про файл
`imshow(...)`, `image(...)`, `imagesc(...)`	відображення масивів
`colorbar`	відображення шкали кольорів
`mat2gray(...)`, `im2uint8(...)`, `im2uint16(...)`,	функції перетворення типу
`IMAGE_PATH(...)`	читання та зміна шляхів до графічних файлів
`size(...)`, `numel(...)`	визначення розміру та кількості елементів масиву
`colormap(...)`	вибір палітри
`surface(матриця,...)`, `surf(матриця,...)`	відображення матриці у вигляді поверхні
`mesh(матриця,...)`	побудови каркасного зображення
`axis equal`, `axis image`	налаштування осей
`min`, `max`	пошук максимуму і мінімуму масива
`zeros(m,n)`, `ones(m,n)`, `rand(m,n)`	генерування масива
`cat(i,...)`	конкатенація масивів

Зробимо ще декілька зауважень про роботу середовища GNU Octave.

Для створення додаткового графічного вікна використовується команда `figure`. Наприклад, інструкція

`figure, imshow(f)`
намалює зображення `f` в новому графічному вікні. Воно буде активним і наступні графічні функції будуть працювати з ним.

Команди `hold on` і `hold off` визначають ваше бажання будувати в активному графічному вікні наступні зображення поверх попередніх (`hold on`), або з заміною попередніх графіків (`hold off`).

Команда `close all` закриває всі відкриті графічні вікна.

Команда `clear` видаляє всі змінні з робочого простору Workspace.

Щоб запитати систему про якусь функцію (тобто отримати справку по функції) можна виконати команду `help ім'я_ функції`. Наприклад

```
help imshow
```

або `doc ім'я_ функції`.

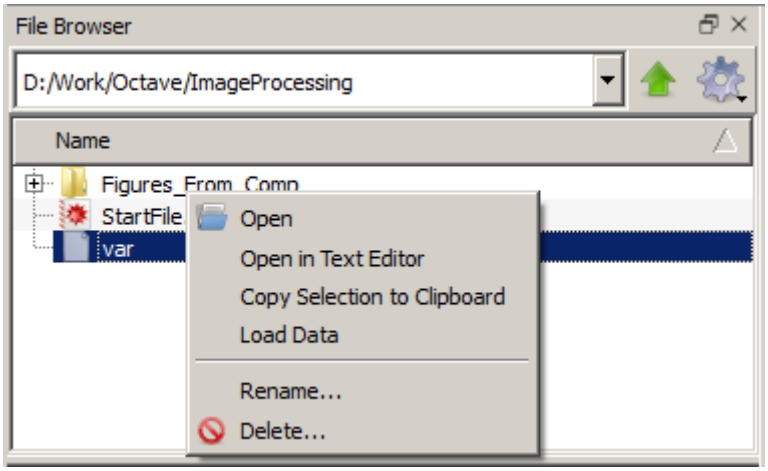
```
doc image
```

яка відкриває більш повну сторінку документації по функції.

Щоб в наступному сеансі виконання середовища GNU Octave продовжити роботу з тими ж змінними, що і в поточному, треба зберегти робочий простір (workspace). Це можна виконати командою

`save ім'я_файла`

Для завантаження збереженого робочого простору необхідно у вікні `File Browser` клацнути правою кнопкою миші по значку файла і у випадному меню обрати команду Load Data.



Ви отримаєте той же результат, якщо виконаєте команду

`load ім'я_файла`

В ім'я файла можна включати шлях до папки його розташування. Якщо шлях не буде вказано, то файли зберігаються в поточному каталозі.

Вище ми розглянули досить прості приклади, для виконання яких можна набирати інструкції в командному рядку. Для більш складних задач кількість інструкцій зростає, і робота в командному рядку стає непродуктивною. Ефективне рішення полягає в оформленні власних алгоритмів у вигляді програм (М-файлів), які можна запустити з робочого середовища або з редактора. Існує два види М-файлів:

- сценарії, які не мають вхідних і вихідних аргументів; вони використовують дані (змінні) з робочого простору;

- функції, які мають вхідні і вихідні аргументи; вони використовують локальні змінні.

Для створення М – файлів в GNU Octave є вбудований редактор.

Коли викликається сценарій, Octave просто виконує команди, що містяться в файлі. Сценарії беруть дані з робочого простору і створюють змінні, які залишаються в робочому просторі.

Наведемо сценарій, який при виконанні прикладів цього параграфа зручно використовувати перед кожним сеансом роботи з Octave.

```
% start file
pkg load image
s=IMAGE_PATH();
IMAGE_PATH(strjoin({s, "D:/Work/Octave/Pictures"}, pathsep));
```

Тобто для роботи з зображеннями ми завантажили пакет розширення 'image' та встановили шлях на папку з файлами зображення.

Функції (або файл-функції) - це М – файли, які можуть мати вхідні аргументи і повертати значення. Ім'я М – файлу і функції повинно бути однаковим. Функції працюють зі змінними в межах власного робочого простору, відмінного від того з яким ви оперуєте в командному рядку.

Щоб при визові сценаріїв та функцій не використовувати шляхи доступу до їх файлів, ці шляхи треба помістити в змінну середовища path. Для цього треба виконати команду addpath(шлях_до файла). Наприклад

```
addpath('D:\Work\Octave\ImageProcessing')
```

Щоб перевірити шляхи доступа виконайте команду path.

Щоб зберегти список шляхів до наступного сеансу роботи, використовується команда savepath.

Інколи вам буде потрібно підключити/встановити до Octave новий пакет розширення. Це виконується командою

```
pkg install -forge package_name
```

Після цього пакет розширення можна загрузити командою

```
pkg load ім'я_пакета
```

Щоб вийти з Octave, наберіть quit або exit в командному рядку.

2. Перетворення яскравості цифрових зображень.

2.1. Перетворення яскравості монохромних зображень.

Результатом дискретизації і квантування монохромного зображення є матриця чисел. Як наслідок, цифрове зображення складається з скінченної кількості елементів, кожний з яких розташований в конкретному місці і має певне значення. Ці елементи прийнято називати пікселями.

Припустимо, що зображення $f(x, y)$ після дискретизації представлено у вигляді матриці, яка складена з М рядків і N стовпців. Тоді кажуть, що зображення має розмір M x N. Координатами лівого верхнього кута зображення вважається пара чисел (1,1). Наступна точка в першому рядку зображення має

координати (1,2). Це, однак, не позначає реальні фізичні координати. Фактично точки зображення ми визначаємо їх індексами (i,j), наприклад, i - номер рядка з відліком від одиниці, а j - номер стовпця. Інколи ми будемо дотримуватися позначення (x, y). Часто індексацію починають з 0, але в Octave індексація масивів починається з 1, тому лівий верхній піксель зображення має координати (1,1), а номери рядків і стовпців змінюються від 1 до M та від 1 до N. Таким чином монохромне цифрове зображення Octave являє собою матрицю

$$f = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1N} \\ f_{21} & f_{22} & \cdots & f_{2N} \\ \vdots & \vdots & & \vdots \\ f_{M1} & f_{M2} & \cdots & f_{MN} \end{bmatrix}.$$

Крім того, ми припускаємо, що дискретні рівні яскравості розташовані з постійним кроком, тобто використовується рівномірне квантування. Інтервал значень яскравості $(f_{\min}, f_{\max})$, де $f_{\min}, f_{\max}$ - мінімальне і максимальне значення матриці f, називають динамічним діапазоном зображення. Якщо значна кількість пікселів займає суттєву частину всього діапазону рівнів сірого, то ми кажемо, що зображення має високий контраст. Навпаки, зображення з малим динамічним діапазоном зазвичай виглядає тьмяним, розмитим і сірим. Збільшення динамічного діапазону зображення робить його більш контрастним.

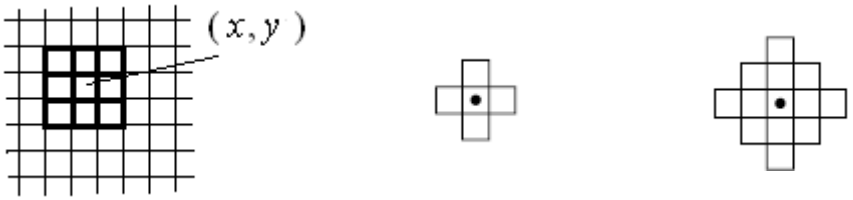
Методи, які ми будемо розглядати в цьому розділі, спрямовані на поліпшення зображень. При цьому слід мати на увазі, що візуальне оцінювання якості зображень є досить суб'єктивним.

Алгебраїчні процедури обробки зображення оперують безпосередньо з пікселями. Методи обробки зображення, що використовуються в цьому розділі, можна описати перетворенням

$$g(x, y) = H[f(x, y)], \qquad (1)$$

де $f(x, y)$ - вхідне зображення, $g(x, y)$- вихідне (оброблене) зображення, а H - деякий оператор (перетворення) над f. Зазвичай цей оператор визначений в деякому околі точки (x,y). У ряді випадків він може обробляти кілька зображень, наприклад, може підсумовувати або усереднювати кілька вхідних зображень.

Основа алгебраїчних методів обробки зображення полягає у визначені просторового околу навколо точки (x,y), який може бути квадратною, прямокутною або іншою по формі ділянкою з центром в точці (x,y), як показано на наступному малюнку.



Центр заданого шаблонного околу (x, y) зміщується від пікселя до пікселя, починаючи з верхнього лівого кута, і на своєму шляху накриває різні ділянки. В

процесі обчислень використовуються тільки пікселі всередині цієї ділянки. Перетворення H застосовується в кожній точці (x, y), даючи в результаті вихідне (оброблене) зображення $g(x, y)$.

Найпростіша форма оператора H виходить, коли окіл має розмір `1 x 1`, тобто складається з одного пікселя. В цьому випадку значення g в точці (x, y) залежить тільки від значення `f` в цій точці, і `H` стає функцією, яка називається перетворенням яскравості/контрастності.

$$g(x, y) = h(f(x, y)). \qquad (2)$$

В залежності від задачі функцію $h(r)$ можна обирати по-різному.

Приклад 1. *Посилення контрастності*. На наступному рисунку зліва показано вхідне зображення `f`. Застосовуючи до нього перетворення

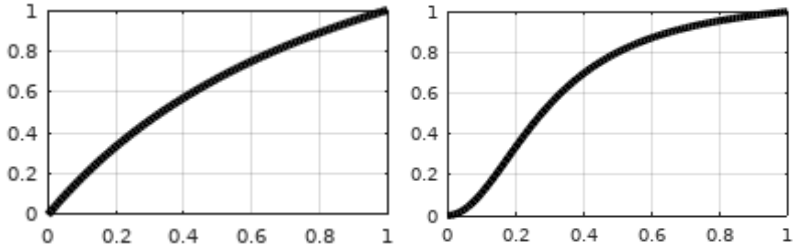
$$g_1 = \frac{2f}{f+1} \quad , \quad g_2 = \frac{12f^2}{11f^2+1},$$

ми отримуємо зображення, наведені в центрі і справа. Але спершу ми виконуємо претворення матриці `f` з типу `uint8` до матриці `g` типу `double`.

```
f=imread('Bone.tif');
g=mat2gray(f);
g1=2*g./(g+1);
g2=12*g.^2./(11*g.^2+1);
subplot(1,3,1),imshow(f);
subplot(1,3,2),imshow(g1);
subplot(1,3,3),imshow(g2);
```

Криві, що використовуються для отримання середнього і правого зображення, і виконують перетворення яскравості, мають наступні графіки

```
x=0:0.01:1; y=2*x./(x+1);
plot(x,y,'k','LineWidth',4); grid on;          % лівий графік
y=12*x.^2./(11*x.^2+1);
figure,plot(x,y,'k','LineWidth',4); grid on;   % правий графік
```



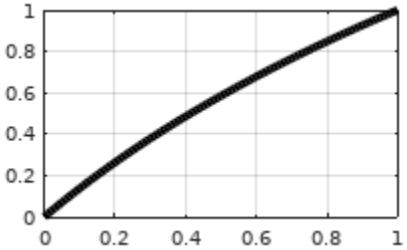
Відкладаючи на горизонтальній осі значення яскравості точки зображення `f`, яке в нашому прикладі належить діапазону `[0 1]`, на вертикальній осі знаходимо значення перетвореної яскравості зображень g_1 і g_2.

■

Функції $h(r)$, які виконують перетворення (2), можуть вибиратися по-різному. Інколи використовується перетворення за допомогою степеневої та логаріфмічної функції.

$$h = c \cdot \ln\left(1 + \alpha r^k\right) \qquad (3)$$

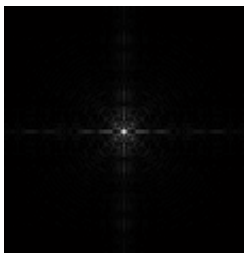
де c, α, k - деякі константи. Форма кривої (3) при $c = 1/\ln 2$, $\alpha = 1, k = 1$ показана на наступному рисунку.



Для реального зображення елементи матриці *r* мають бути перетворені до дійсного діапазону `[0 1]`.

Приклад 2. Використання перетворення (3).

```
f=imread('Spectrum.jpg');          % тип uint8
imshow(f);
```

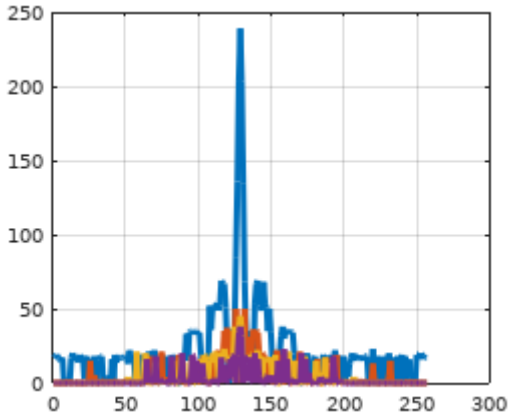


Дізнаємося розмір зображення і знайдемо максимальне значення в масиві `f`.

```
size(f)
ans = 257   257
max(max(f))
ans = 254
```

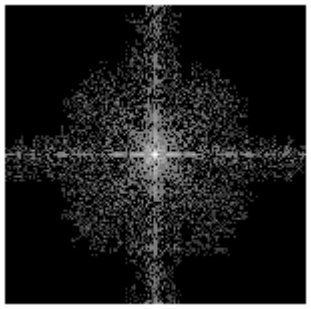
Побудуємо графіки перерізу матриці вздовж деяких горизонтальних прямих.

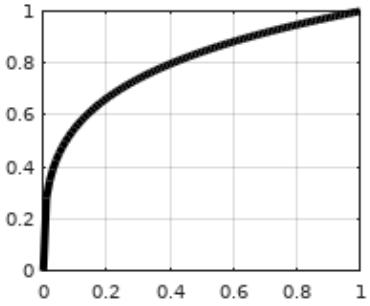
```
x=[1:1:257];
figure,plot(
  x,f(128,:),'LineWidth',3,...
  x,f(120,:),'LineWidth',3,...
  x,f(110,:),'LineWidth',3,...
  x,f(100,:),'LineWidth',3);
```



Як бачимо, велика частина зображення має значно менші значення яскравості, ніж в точці максимуму. Тому як сама яскравість мала, так і різниця в яскравості точок, розташованих далеко від центру мала і практично непомітна. Спробуємо застосувати перетворення (3).

```
g=mat2gray(f);
g1=log(1+g.^(1/3))/log(2);figure,imshow(g1);  % зображення ліворуч
```





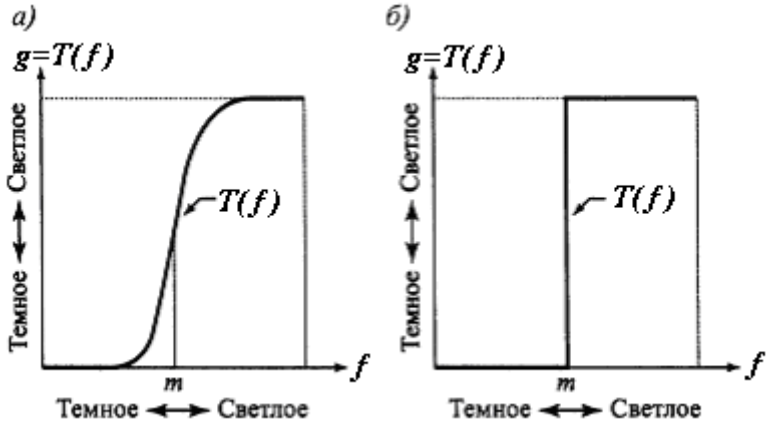
Тут ми ще використали функцію `mat2gray`, яка переводить елементи матриці `f` до діапазону `[0 1]`.

Графік функції перетворення показано на попередньому рисунку праворуч. Його побудовано за допомогою наступного коду.

```
x=[0:0.01:1];
y=log(1+x.^(1/3))/log(2);
figure, plot(x,y,'k','LineWidth',4); grid on;  % зображення праворуч
```

■

Функція, графік якої показано на наступному рисунку зліва, називається функцією перетворення контрастності, оскільки вона розтягує вхідні величини, менші ніж `m`, в більш широкий піддіапазон темних рівнів на вихідному зображенні, і, відповідно, величини, більші `m`, - в більш широку смугу яскравих рівнів. В результаті, якщо значна частина пікселів має яскравість в околі `m`, то створюється зображення з більшою контрастністю.



У граничному випадку, показаному на попередньому рисунку праворуч, виходом служить двійкове (чорно-біле) зображення. Відповідна функція називається пороговим перетворенням.

Існує багато способів побудувати функцію, профіль якої схожий на останній наведений ліворуч графік. Наприклад, рівняння схожої кривої можна записати у вигляді

$$g = H(f) = \frac{1}{1+(m/f)^q} \qquad (3)$$

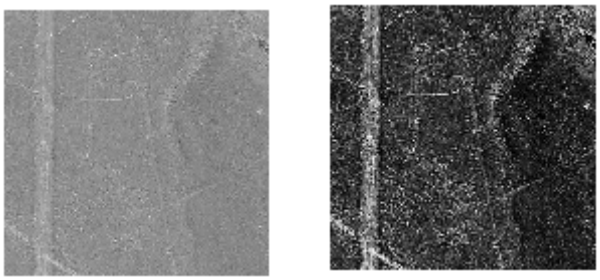
де `f` - це яскравість пікселів вхідного зображення, `g` - відповідна яскравість вихідного зображення, а параметр `q` контролює нахил кривої. Це рівняння реалізується в Octave у вигляді такої інструкції

```
g = 1./(1 + ( m./(double(f) + eps)).^q)
```

Використання величини `eps` (машинного епсілон) дозволяє уникнути помилки переповнення, якщо в `f` є нульові елементи. Оскільки верхнє граничне значення функції `H(f)` дорівнює 1, вихідні значення масштабовані в діапазоні [0,1].

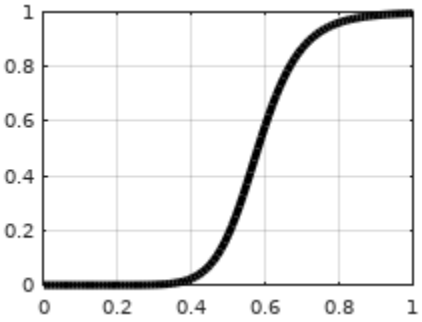
Приклад 2. *Посилення контрастності*. На наступному рисунку зліва показано монохромне зображення `f` геогліфа Наски (Перу). Застосовуючи до нього перетворення (3) при `m=0.58`, `q=10` ми отримуємо зображення, наведене праворуч. Але спершу ми виконали претворення матриці `f` з типу `uint8` до матриці `g` типу `double`.

```
f=imread('Nasky02.bmp');
imshow(f)
g=mat2gray(f);
g2=1./(1+(0.58./(g+eps)).^10);
figure, imshow(g2)
```



Графік кривої перетворення контрастності показано нижче і отримано наступними інструкціями.

```
f=[0:0.01:1];
g=1./(1+(0.58./(f+eps)).^10);
plot(f,g,'k','LineWidth',4); grid on;
```



■

Кількість функцій, які мають схожі профілі, дуже велика. Ми лише наведемо приклади декількох з них.

Найпростіший тип – це ламана, якщо слушно підібрати координати її вузлів.

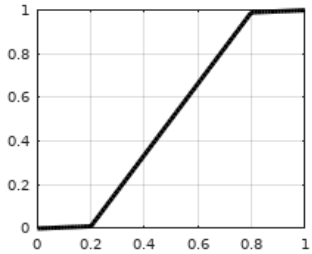
Для обробки зображення за допомогою ламаної ми створили процедуру `brokenline`, яка генерує координати точок ламаної по координатам її вузлів. Код функції наведено в додатку А.

Приклад 3. *Посилення контрастності за допомогою ламаної.*

```
f=imread('Univer02.jpg');
g=mat2gray(f);
imshow(g)
pnt=[0 0 0.2 0.01 0.8 0.99 1 1];
g2=brokenline(g,pnt);
figure,imshow(g2);
```

Графік кривої перетворення контрастності/яскравості показано нижче.

```
x=0:0.01:1; y=brokenline(x,pnt);
plot(x,y,'k','LineWidth',3); grid on;
```



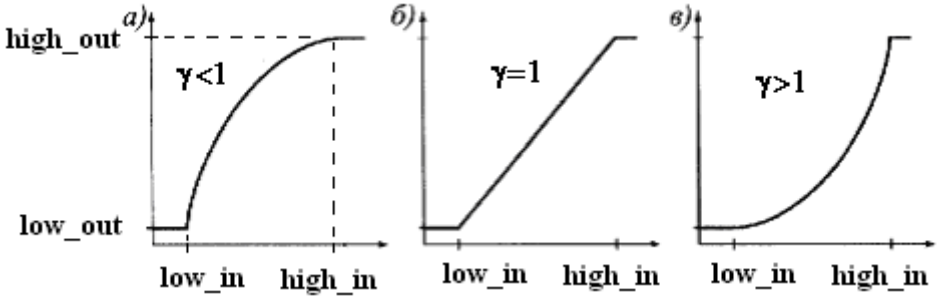
■

Вибирати параметри функцій перетворення контрастності іноді буває складно. Для полегшення такого підбору в пакеті `'image'` є досить універсальна функція `imadjust`. При перетворенні яскравості півтонових зображень вона має наступний синтаксис.

`g=imadjust(f,[low_in;high_in],[low_out;high_out],gamma)`
Функція конвертує зображення `f` в нове зображення `g`, при якому значення яскравості в інтервалі `[low_in; high_in]` переходять в інтервал `[low_out; high_out]`. Значення, менші за `low_in` відображається в `low_out`, а все, що більше `high_in` відображається в `high_out`.

Вхідне зображення може мати клас `uint8, uint16` або `double`, і клас вихідного зображення збігається з класом вхідного. Всі інші вхідні параметри функції `imadjust` повинні бути дійсними числами в інтервалі від 0 до 1, незалежно від класу `f`. Якщо замість векторів `[low_in; high_in]` або `[low_out;high_out]` поставити порожній вектор (`[]`), то будуть використовуватися величини за замовчуванням `[0 1]`. Якщо `high_out` менше, ніж `low_out`, то вихідні яскравості симетрично перевертаються (виходить негатив).

Параметр `gamma` служить для задання форми кривої, яка відображає яскравість `f` в яскравість `g`. Якщо `gamma` менше 1, то яскравість відображення зміщується вгору в сторону більш яскравих значень, утворюючи опуклу вгору криву (наступний рисунок *а*). Якщо `gamma` більше 1, то яскравість відображення зміщується вниз в сторону менш яскравих значень (*в*). Якщо параметр `gamma` опущений, то його значення за замовчуванням дорівнює 1 (лінійне відображення *б*).



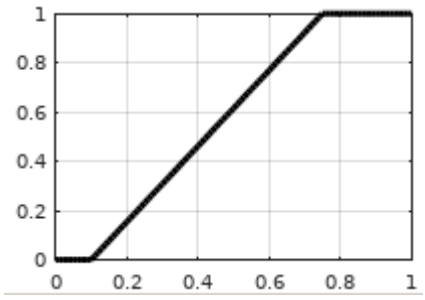
Приклад 4. *Використання функції* `imadjust`.

```
f=imread('Moon2.jpg');
f=mat2gray(f);
size(f)
ans =   540   466
imshow(f);   % наступний рисунок ліворуч
g1 = imadjust(f, [0.1 0.75], [0 1]);
figure,imshow(g1);  % зображення всередині
```

Графік функції перетворення показано праворуч.

```
x=0:0.01:1; y=imadjust(x, [0.1; 0.75], [0 1]);
figure, plot(x,y,'k','LineWidth',3); grid on;  % зображення праворуч
```





Процедура `imadjust(f,[0 1],[1 0])` є цифровим еквівалентом отримання фотонегативів.

```
g2 = imadjust(f, [0 1], [1 0]);
figure,imshow(g2);   % наступне зображення ліворуч
```


Негативне зображення можна побудувати також за допомогою функції `imcomplement`.

```
g3 = imcomplement(f);
figure,imshow(g3);
```

Результатом буде негативне зображення місяця,таке саме, як на попередньому рисунку ліворуч.

Можна побудувати негатив з меншими відтінками сірого. На попередньому рисунку праворуч наведені результати виконання команд

```
g4 = imadjust(f, [0.25 0.75], [1 0]);
figure,imshow(g4);
```

які розтягують шкалу градації між 0.25 і 0.75 на весь інтервал [0,1].

Нарешті, по команді

```
g5 = imadjust(f, [ ], [ ] , 0.3) ;
figure,imshow(g5);   % наступний рисунок ліворуч
```

створюється зображення з великою кількістю сірих тонів.


Зверніть увагу, що на попередньому рисунку ліворуч в околі затінка з’явилася смуга, яка не проглядалася на інших зображеннях.

Можна також побудувати негатив з меншими відтінками сірого

```
g6 = imadjust(f, [ ], [1 0] , 0.7) ;
figure,imshow(g6);   % попередній рисунок праворуч
```

■

Зауваження. Функція `imadjust(f,...)` не завжди коректно працює з параметром $\gamma \neq 1$. Тому ми не будемо наводити відповідні приклади. Протестуйте роботу функції для цих випадків самостійно.

Попередні два приклади будували ламану (функцію перетворення яскравості) з двома вузлами. Але наша функція `brokenline` дозволяє створювати ламані з будь якою кількість вузлів, і отже має більшу гнучкість. До того ж, її лівий і правий прямолінійні відрізки необов'язково повинні бути горизонтальними.

Приклад 5. На наступному рисунку ліворуч наведено вхідне малоконтрастне 8-бітове зображення – фотографія пилку, яка зроблена на електронному мікроскопі.

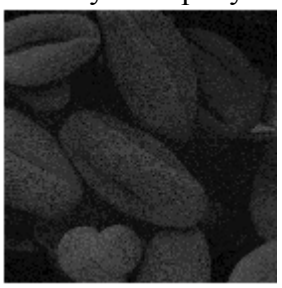
```
I=imread('Pollen2.jpg'); imshow(I);    % наступний рисунок ліворуч
```

Оскільки

```
max(max(f))
ans = 94
```

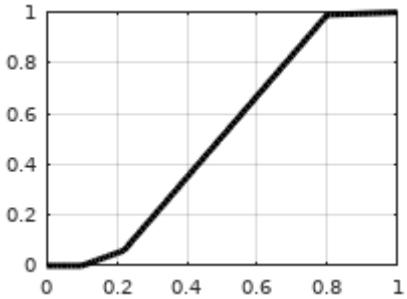
то вже функція `mat2gray` покращить зображення, окільки вона перетворить шкалу яскравостей $0 \le I \le 94$, яка займає частину діапазону [0 255], на весь діапазон яскравостей $0 \le F \le 1$.

```
F=mat2gray(I);
figure,imshow(F);  % наступний рисунок праворуч
```




На наступному рисунку ліворуч показано графік ламаної, яка використана для підвищення контрасту зображення F, і результат її застосування.

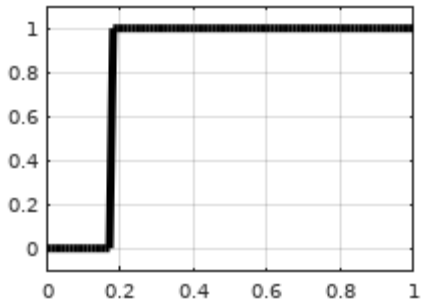
```
x=0:0.01:1;
pnt=[0 0 0.1 0 0.22 0.06 0.8 0.99 1 1];
y=brokenline(x,pnt);
figure, plot(x,y,'k','LineWidth',3); grid on;  % наступний рисунок ліворуч
G=brokenline(F,pnt);
figure,imshow(G)  % наступний рисунок праворуч
```



Результат порогового перетворення показаний нижче.

```
g1=(F>0.5); figure,imshow(g1);      % наступний рисунок ліворуч
x=0:0.01:1;figure,plot(x,x>0.175,'k','LineWidth',4);  % наступний рисунок праворуч
```

```
grid on; axis([ 0 1 -0.1 1.1]);
```

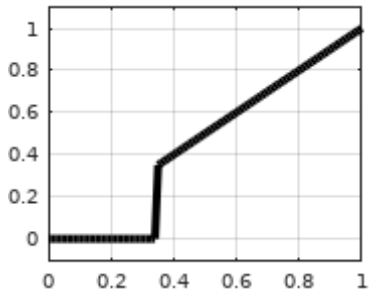


Порогове перетворення можна комбінувати з початковим зображенням F. Наприклад формула `F.*(F>0.35)` створює зображення, в якому пікселі з яскравістю меншою за 0.35 стають чорними (яскравість 0), а яскравість інших залишається незмінною.

```
g2=F.*(F>0.35);  figure,imshow(g2);  % наступний рисунок ліворуч
```

Це еквівалентно перетворенню

```
x=0:0.01:1;figure,plot(x,x.*(x>0.35),'k', 'LineWidth',4); % наступний графік праворуч
grid on; axis([ 0 1 -0.1 1.1]);
```

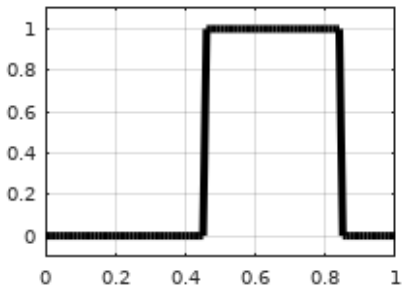


Використання двобічної функції – сходинки створює наступне зображення

```
g3=((f>0.45) & (f<0.85));
figure,imshow(g3);                  % наступний рисунок ліворуч
```

Це еквівалентно перетворенню, графік якого показано на натупному рисунку праворуч.

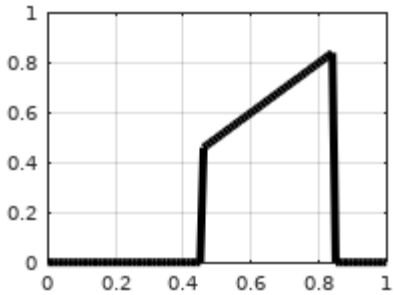
```
x=0:0.01:1;y=((x>0.45) & (x<0.85));
figure,plot(x,y,'k', 'LineWidth',4);   % наступний графік праворуч
grid on;axis([ 0 1 -0.1 1.1]);
```

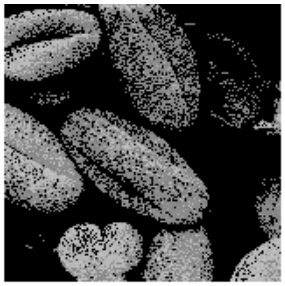


Використання поелементного добутку двобічної функції–сходинки з матрицею початкового зображення, створює наступні функцію перетворення і відповідне зображення.

```
x=0:0.01:1;
figure,plot(x,x.*((x>0.45) & (x<0.85)),'k', 'LineWidth',4); % наступний графік ліворуч
```

```
grid on;
g4=f.*((f>0.45) & (f<0.85));figure,imshow(g4); % наступний рисунок праворуч
```





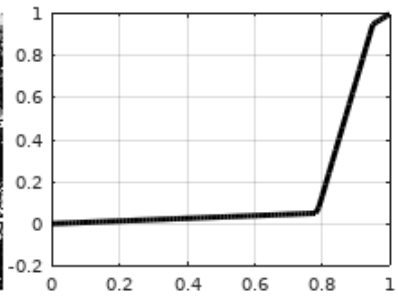
Приклад 6. Ось ще приклад застосування кусково - лінійного перетворення, який використовує функцію `brokenline`.

```
f=imread('Aero2.jpg');
max(max(f))
ans = 255
f=mat2gray(f);
imshow(f);                  % наступний рисунок ліворуч
pnt=[0 0 0.785 0.05 0.95 0.95 1 1];
g=brokenline(f,pnt);
figure,imshow(g)            % наступний рисунок всередині
```

Як бачимо, контраст зображення збільшився. Праворуч побудовано графік вікористаного перетворяння.

```
x=0:0.01:1; y=brokenline(x,pnt);
figure, plot(x,y,'k','LineWidth',3); grid on;  % наступний рисунок праворуч
```



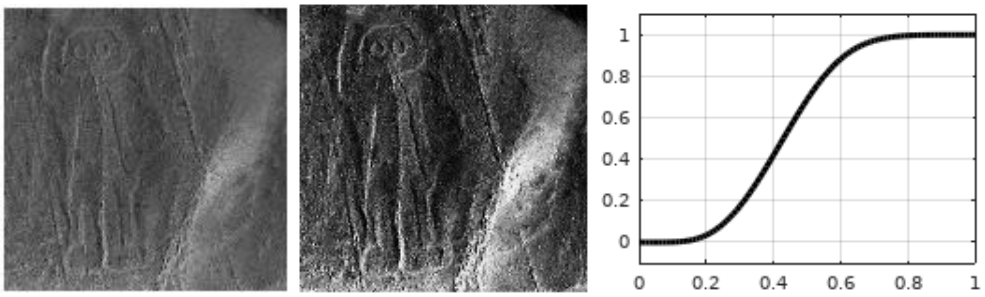


■

Замість функцій типа формули (3) можна використовувати сплайни–сходинки. Код такої функції наведено в додатку Б.

Приклад 7. *Застосування сплайн–сходинок* в якості функції перетворення яскравості.

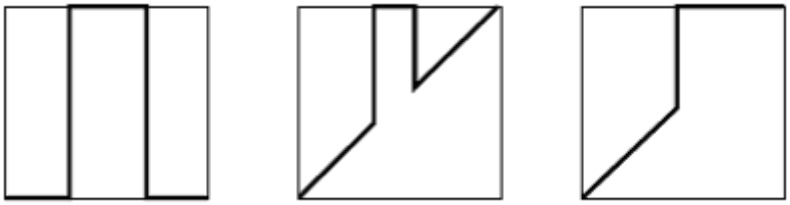
```
f=imread('Nasky03.png');
f=mat2gray(f);
imshow(f);                  % наступний рисунок ліворуч
g=stepspline(f,[0 0.1 0.2 0.6  0.7 1]);
figure,imshow(g);           % наступний рисунок всередині
x=0:0.01:1;y=stepspline(x,[0 0.1 0.2 0.6  0.7 1]);
figure, plot(x,y,'k','LineWidth',3);  % наступний рисунок праворуч
grid on;   axis([0 1 -0.1 1.1]);
```



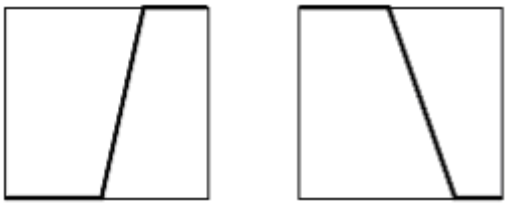
■

Функції поелементних перетворень яскравості можуть мати різну форму. Перелічимо деякі інші види таких перетворень.

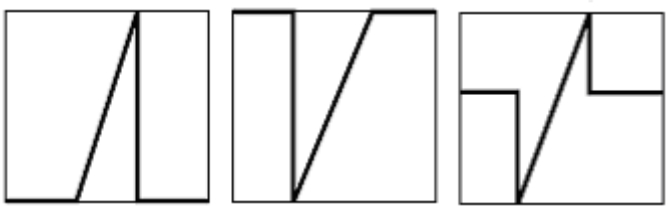
Узагальненням порогового перетворення є перетворення зрізу (наступний графік зліва). Воно дозволяє виділити певний інтервал діапазону яскравості вхідного зображення. Переміщаючи «робочий» інтервал за шкалою і змінюючи його ширину, можна здійснювати візуальний аналіз об'єктів, що розрізняються по яскравості. Деталі, що не потрапили в обраний інтервал, будуть придушені. На наступному малюнку всередині наведено графік функції перетворення зрізу зі збереженням фону. У цьому випадку зображення в цілому зберігається, але на ньому "висвічуються" ділянки, що потрапили в заданий інтервал яскравостей. Якщо цей інтервал примикає до границі шкали яскравості, то отримуємо перетворення, зване неповної порогової обробкою (графік праворуч).



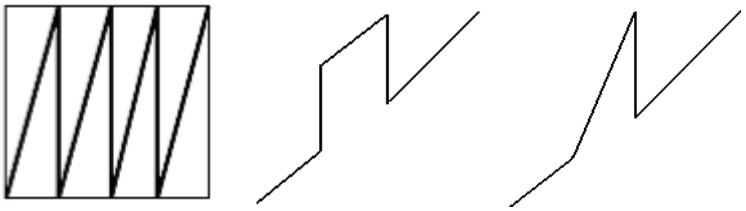
Контрастне масштабування, графік якого показаний на наступному рисунку ліворуч, нами вже розглядався. Тут «робочий» інтервал яскравості розтягується на весь діапазон можливих значень. Контрастне масштабування, яке відповідає графіку праворуч, обертає функцію яскравості, тобто дає негатив.



Контрастне масштабування, графіки якого показані на наступному рисунку, виділяє «робочий» діапазон яскровості на чорному фоні (наступний рисунок ліворуч), на білому фоні (рисунок в середині), на сірому фоні (графік праворуч).



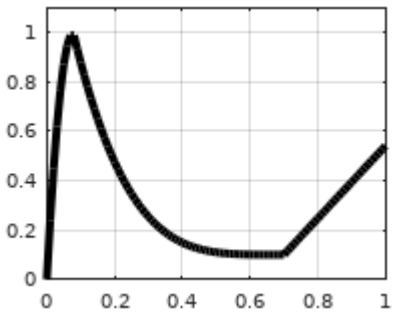
Пилкоподібне контрастне масштабування (наступний рисунок ліворуч) зручно застосовувати для зображень, що складаються з декількох областей з повільно мінливим значенням яскравості. Таке перетворення майже не руйнує цілісності сприйняття зображення, але різко збільшує контрастність погано помітних дрібних деталей. На наступному малюнку в середині і праворуч приведені ще два графіка можливих корисних перетворень.



Можна створювати кускові перетворення більш витонченої форми. Наведемо один такий приклад з кусково заданою функцією перетворення.

Приклад 8.

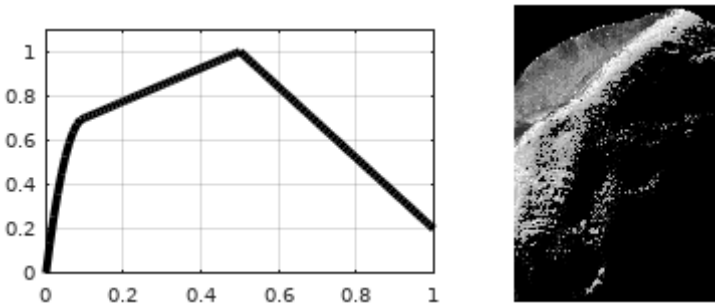
```
close all;
% Функція перетворення
a=0.078;
z=@(x) x.*(2*a-x)/a^2.*(x<a)+
       (((x-0.7).^4/0.6386^4)+0.1).*(x>=a & x<=0.7) +
                                    ((x-0.7)/0.68+0.1).*(x>0.7);
x=0:0.01:1;
plot(x,z(x),'k','LineWidth',4); % наступний графік ліворуч
grid on; set(gcf,'Color',[1 1 1]); axis([0 1 0 1.1]);
% вихідне зображення
f=imread('Phobos2.jpg');
f=mat2gray(f);
figure, imshow(f);  % наступний рисунок всередині
g=z(f);
figure, imshow(g);  % результівне зображення, наступний рисунок праворуч
```



Замініть функцію z

```
z=@(x) (0.7.*x.*(2*a-x)./a.^2).*(x<a)+ ...
     (0.7+3/4*(x-0.1)).*(x>=a & x<=0.5) + ...
     (1-8/5*(x-0.5)).*(x>0.5);
```

і ви отримаєте наступну криву перетворення яскравості і відповідне зображення.

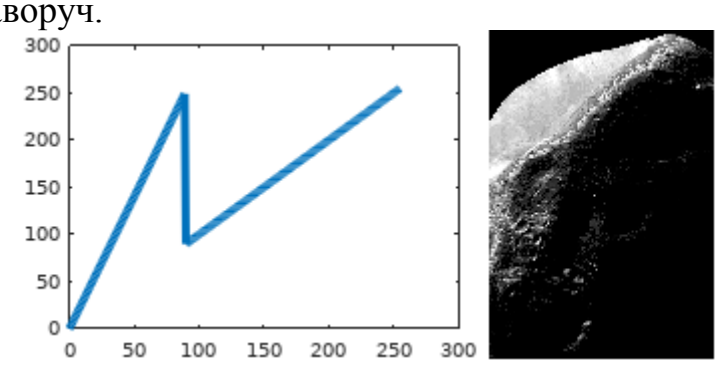


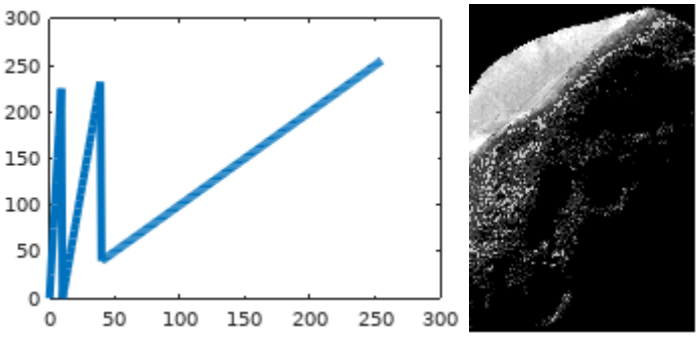
Спробуйте ще перетворення `g=2*(0.5-abs(f-0.5))`.

Ось ще два перетворення цього зображення. Вони побудовані наступним сценарієм.

```
close all
f=imread('Phobos2.tif');
x=0:1:255;

function z=f1(x)
  z=(x<90).*x*2.8+(x>=90).*x;
endfunction

function z=f2(x)
  z=(x<10).*x*25+(10<=x & x<40).*(x-10)*8+(x>=40).*x;
endfunction

figure, plot(x,f1(x),'LineWidth',4);
g1=f1(f); figure, imshow(g1);
figure, plot(x,f2(x),'LineWidth',4);
g2=f2(f); figure, imshow(g2);
```

Графіки перетворень показано на наступних рисунках ліворуч, а перетворені зображення – праворуч.





У цьому прикладі ми змогли виділити в темній частині фотографії деякі деталі, які не проглядалися на початковому зображенні. Зверніть увагу на спосіб

створення кусочних функцій. Його особливістю є використання логічних виразів типу `(x<0.078)` або `(x>=0.078 & x<=0.7)`, які для кожного елемента вектора або матриці `x` набувають значення нуль або одиниця.

■

Вибір відповідного перетворення не завжди є простою задачею. Іноді в цьому нам може допомогти гістограма зображення.

Гістограмою цифрового зображення називається дискретна функція

$$h(r_k) = n_k$$

де r_k – це k-й рівень яскравості (з інтервалу квантування яскравостей `[0, R]`), а n_k – кількість пікселів зображення, рівень яскравості яких дорівнює r_k. Значення `R` дорівнює 255 для зображень класу `uint8` і 65535 - для класу `uintl6`.

Головною функцією пакета `'image'` для поводження з гістограмами є функція `imhist` з наступним синтаксисом:

```
h=imhist(f,b),
```

де `f` - це вхідне зображення, `h` - його гістограма $h(r_k)$ (фактично вектор значень), і `b` - кількість кошиків, використаних при формуванні гістограми (якщо аргумент `b` відсутній, то за замовчуванням береться `b=256`). Кошиком називається підрозділ шкали яскравостей. Наприклад, якщо при роботі з зображеннями класу `uint8` змінна `b = 2`, то шкала яскравості ділиться на дві зони (кошика): від 0 до 127 та від 128 до 255. Підсумкова гістограма буде мати два значення: `h(1)`, яка дорівнює кількості пікселів зображення, величини яких перебувають в інтервалі [0,127], і `h(2)`, яке дорівнює кількості пікселів зі значеннями в інтервалі [128, 255].

Використання функції `imhist(...)` без повертання значення призводить до негайної побудови гістограми.

Щоб отримати нормовану гістограму, треба виконати ділення

```
p = imhist(f, b)/numel(f),
```

де `numel(f)` це кількість елементів масиву f, тобто кількість пікселів зображення.

Приклад 8. *Обчислення гістограм і побудова їх графіків.*

```
f=imread('Aero2.tif');
imhist(f);  % гістограма за замовчуванням
```

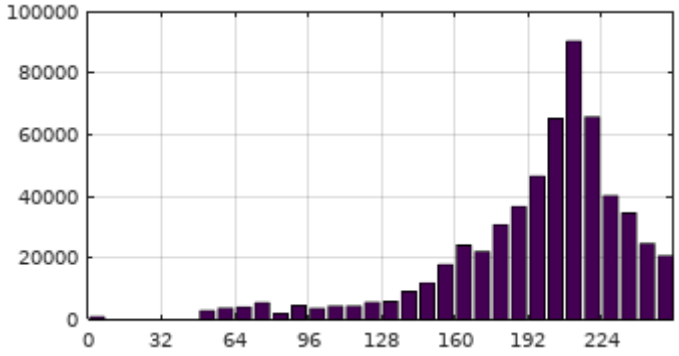
Є багато інших способів побудови гістограм.

Гістограму можна побудувати за допомогою стовпчастих діаграм. Для цього призначена функція `bar(X,Y,W)`, де `Y` - це вектор-рядок, графік якого треба побудувати, `X` - вектор того ж розміру, що і вектор `Y`, який представляє горизонтальні координати розташування стовпчиків вектора Y. Якщо параметр

X опущений, то горизонтальна вісь ділиться з кроком 1 від 0 до length(Y). Параметр W - число між 0 і 1. Коли W дорівнює 1, сусідні стовпчики стикаються, коли W близько до 0 (наприклад, 0.01) , стовпчики є вертикальними лініями, як на попередньому малюнку. За замовчуванням W дорівнює 0.8.

Наступні команди призводять до побудови стовпчастої діаграми, на якій горизонтальна вісь ділиться на 32 кошики.

```
close all;
f=imread('Aero2.tif');
Y=imhist(f,32);
X = 4:8:256;
figure, bar(X, Y); grid on;
axis([0 255 0 100000]);
set(gca, 'xtick', 0:32:255)
set(gca, 'ytick', 0:20000:100000)
```



Функція axis має синтаксис

```
axis([horzmin horzmax vertmin vertmax])
```

Вона встановлює мінімальні та максимальні значення горизонтальних і вертикальних координат. В останніх двох інструкціях gca означає фразу «get current axis» (використовувати поточні осі, тобто осі щойно побудованого графіка; точніше, функція gca повертає посилання на об'єкт «axis»), а xtick і ytick встановлюють мітки/зарубки на горизонтальній і вертикальної осі в заданих точках.

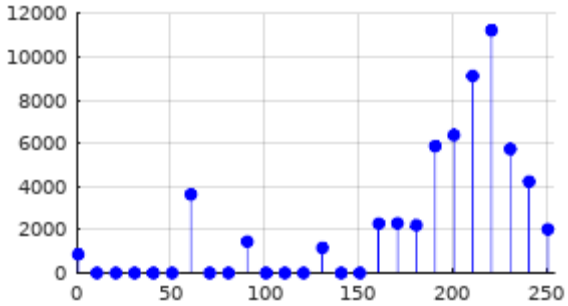
Стеблева діаграма будується аналогічно столбчастій

```
stem(X,Y,'color_linestyle_marker', 'fill'),
```

де Y - це вектор-стовпець, графік якого необхідно побудувати, а змінна X має таке ж значення, що і в функції bar. Аргумент color_linestyle_marker складається з комбінації символів, які керують стилем діаграми. Наприклад, команда stem(Y,'r s') побудує стеблеву діаграму, на якій вертикальні стебла будуть проведені червоними лініями, а головки стебел будуть квадратними. За замовчуванням лінії стебел вважаються суцільними, а головками служать кола. Стеблевая діаграма на наступному малюнку показує кожне десяте значення з діапазону яскравостей і отримана за допомогою такого сценарію.

```
close all;
f=imread('Aero2.tif');
Y=imhist(f);
X = 1:10:256; Y1 = Y(X);
figure, stem(X,Y1,'ob','fill');  grid on;      % 'o' – кружальце, 'fill' - зафарбовані
```

```
axis([0 255 0 12000]);
```

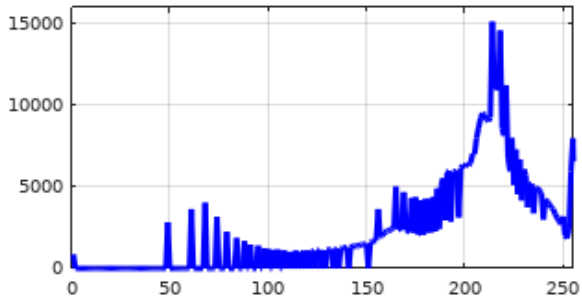


Функція `plot` будує графік по точках, поєднуючи їх відрізками прямих ліній. Вона має синтаксис

`plot(X,Y,color_linestyle_marker')`,

де аргументи мають таке ж значення, що і у функції `stem`.

```
h=imhist(f);  plot(h,'b','LineWidth',3);
axis([0 255 0 16000]); grid on;
```



Повернемося до прикладу 8. Після побудови гістограми ми знаємо, що наше зображення `f` має дуже мало пікселів з яскравістю в діапазоні 0 - 50. Тому можна застосувати перетворення яскравості, з придушенням цих пікселів (зображення в центрі). Залишаючи динамічний діапазон яскравості 150 - 255 ми отримуємо ще більш контрастне зображення (наступний рисунок праворуч)

```
f=imread('Aero2.tif');  imshow(f);
g1=imadjust(f,[0.2 1],[0 1],1); figure, imshow(g1);
g2=imadjust(f,[0.6 1],[0 1],1); figure,imshow(g2);
```



Таким чином, якщо на гістограмі є одна згрупована зона, то для підвищення контрастності зображення треба розтягнути діапазон яскравості, що їй відповідає.

■

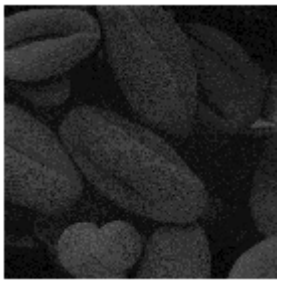
Нехай на початковому зображенні є L градацій яскравості $r_1 < r_2 < ... < r_L$, які зустрічаються відповідно $n_1, n_2, ..., n_L$ раз (тобто значення яскравості r_i мають

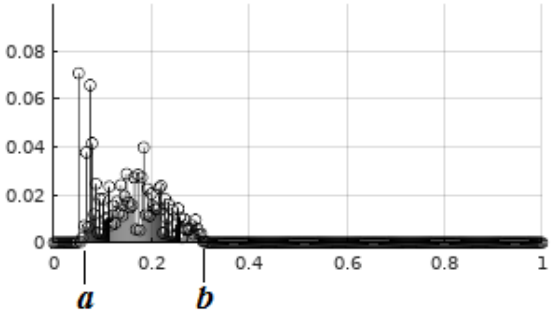
n_i пікселів) і покладемо $n = \sum_{j=1}^{L} n_j$. Перетворимо зображення в такий спосіб – замінимо градації r_i на величини

$$s_i = T(r_i) = \sum_{j=1}^{i} \frac{n_j}{n} \; (i = 1,...,L). \qquad (4)$$

тобто s_i будуть яскравостями пікселів, які мали на початковому зображенні яскравість r_i. При цьому очевидно, що всі $s_i \le 1$, оскільки в сумі (4) стоять тільки додатні елементи і найбільшим значенням може бути $s_L = \sum_{j=1}^{L} \frac{n_j}{n} = \frac{1}{n} \sum_{j=1}^{L} n_j = 1$. Результат перетворення (4), званого еквалізацією зображення, полягає в збільшенні динамічного діапазону рівнів яскравості, що зазвичай означає більшу контрастність вихідного зображення. Дійсно, нехай зображення має чітко виділену зону яскравостей, наприклад, таку, яка показана на нормованій гістограмі на наступному рисунку праворуч.

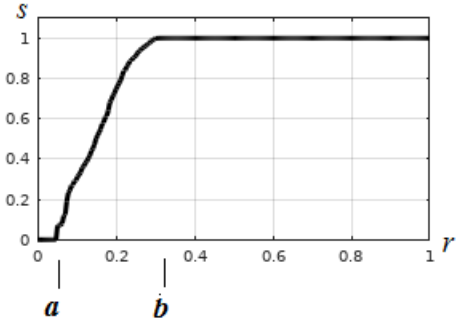
```
close all;
f=imread('Pollen2.tif'); imshow(f);
hn = imhist(f)./numel(f);          % нормування гістограми до діапазону 0 - 1
x = linspace(0,1,256);             % вектор 256 значень від 0 до 1
stem(x,hn,'.k');                   % наступний рисунок ліворуч
axis([0 1 0 0.1]); grid on;
```





Тоді перетворення (4) дасть функцію $s = T(r)$, яка буде нулем (або дуже мала) на ділянці $r < a$, монотонно зростаючою на відрізку $[a\,b]$ і константою (або майже не зростаючою) на ділянці $r > b$. При цьому розтягнення зони яскравості буде якраз в тому місці, де зосереджені основні градації яскравості зображення. Функція перетворення матиме такий вигляд

```
sdf = cumsum(hn);
figure, plot(x,sdf,'k','LineWidth',3);  % наступний графік праворуч
grid on; axis([0 1 0 1.1]);
```



Таким чином, функція (4) перетворює вузький вхідний діапазон яскравості на всю шкалу яскравості вихідного зображення.

Тут ми використали функцію `cumsum` накопичувального підсумовування. Якщо A - вектор довжини N, то `B=cumsum(A)` є вектором елементів B_k, які обчислюються за формулою $B_k = \sum_{i=1}^{k} A_i \ \left(k = 1,2,...,N\right)$. Тобто $B_1 = A_1$, $B_2 = A_1 + A_2$, $B_3 = A_1 + A_2 + A_3$, ... , $B_N = A_1 + ... + A_N$. Якщо A - багатовимірний масив, то `B=cumsum(A,dim)` - накопичувальна сума елементів вздовж розмірності `dim`.

Сама еквалізація (тобто перетворення зображення за формулою (4)) реалізовано в пакеті `'image'` функцією `histeq`, яка має синтаксис

```
g=histeq(f,nlev)
```

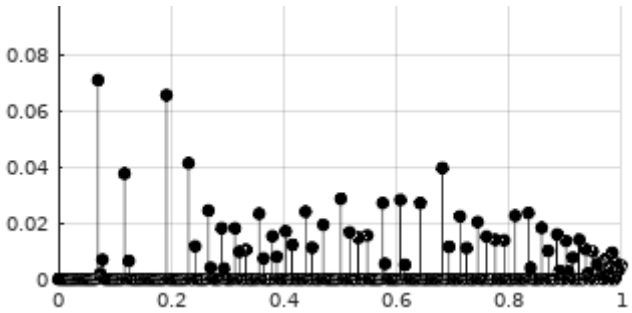
де `f` - це вхідне зображення в `double` форматі зі значеннями в діапазоні від 0.0 до 1.0, а `nlev` - бажана кількість рівнів інтенсивності вихідного зображення. Значенням `nlev` за замовчуванням є 64. Зазвичай ми будемо використовувати максимально можливу кількість рівнів (наприклад, 256) для `nlev`, оскільки це дає справжню реалізацію описаного вище методу гістограмної еквалізациії. Зауважимо, що функція `histeq` з Octave частково відрізняється від однойменної функції Matlab.

Приклад 9. Розглянемо ще раз зображення пилку з файлу `Pollen2.tif` і застосуємо до нього перетворення еквалізації.

```
close all;
f=imread('Pollen2.tif');
imshow(f);
g1=mat2gray(f);
figure,imshow(g1);
g2 = histeq(g1, 256);
figure,imshow(g2);
```

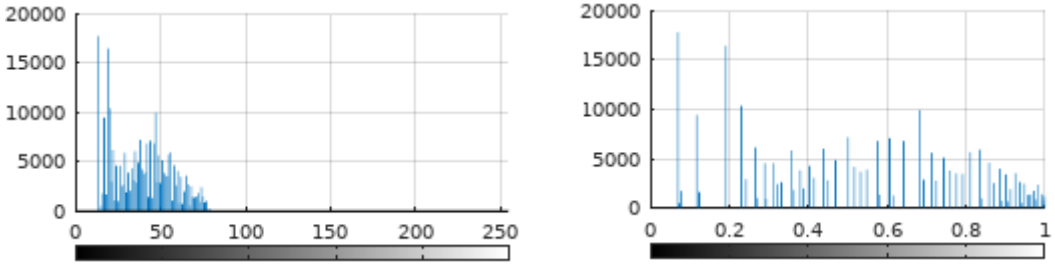
Нормована гістограма початкового зображення і відповідне йому перетворення еквалізаціі показано нами раніше. Нормована гістограма перетвореного зображення має вигляд

```
hnorm = imhist(g2)./numel(g2);
x = linspace(0, 1, 256);
figure,stem(x,hnorm,'ok','fill');
axis([0 1 0 0.1]); grid on;
```



Початкове зображення є дуже темним і гістограма виявляє його темну природу, оскільки вона (гістограма) повністю розташована у темній ділянці шкали яскравості. Також очевидний низький динамічний діапазон зображення, оскільки «ширина» гістограми мала в порівнянні з шириною всього діапазону сірих півтонів. Поліпшення середньої яскравості і контрастності перетвореного зображення очевидно. Вивчивши гістограмму перетвореного зображення, розуміємо, що підвищення контрастності викликано істотним розширенням динамічного діапазону на всю шкалу яскравості. Підвищення загальної яскравості зображення пов'язане з тим, що середній рівень яскравості на гістограмі еквалізованного зображення став вище, ніж на вихідному зображенні. Це можна також побачити порівнявши ненормовані гістограми початкового і перетвореного зображень.

```
figure, imhist(f); grid on;
figure, imhist(g2) ; grid on;
```



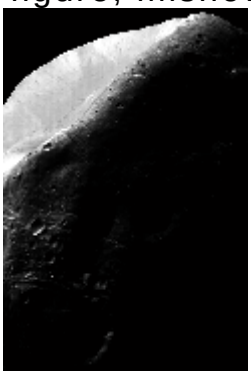
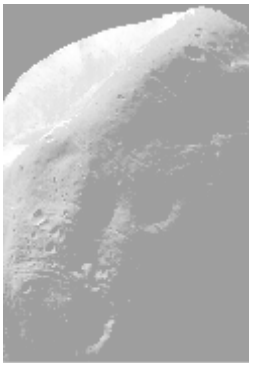
Ось ще один приклад задовільного використання гістограмної еквалізациії.

```
f = imread('pout.tif'); imshow(f);
g=histeq(f); figure,imshow(g);
```

Гістограмного еквалізація відбувається перетворенням, яке залежить від гістограми обробляємого зображення. Тобто функція перетворення не буде змінюватися, доки не зміниться саме зображення. Як зазначалося, гістограмна еквалізація покращує зображення шляхом розширення діапазону його рівнів яскравості. Однак така процедура добре працює тільки для зображень з «вузькою» гістограмою і тому не завжди призводить до задовільного

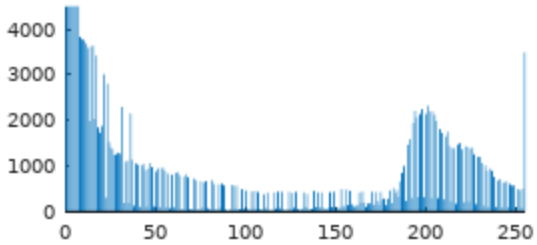
результату. Так для наступного зображення гістограмна еквалізація не дає істотного поліпшення.

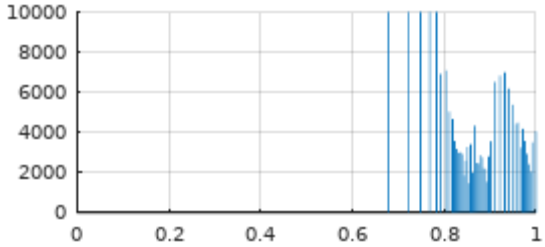
```
f=imread('Phobos2.tif'); imshow(f);      % Фобос (ліворуч)
f1=histeq(f, 256); figure, imshow(f1);  % еквалізація (праворуч)
```

На початковому зображенні домінують темні області, що відповідає великій концентрації темних пікселів. Здавалося б, що гістограмна еквалізація здатна поліпшити якість цього зображення, оскільки деталі темних областей стали б більш помітними. Однак істотного поліпшення зображення не відбулося. Причину цього можна пояснити, аналізуючи гістограму еквалізованного зображення, наведеного на наступному рисунку праворуч.

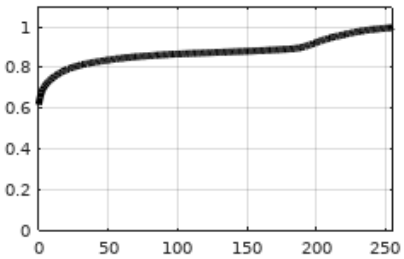
```
figure, imhist(f);
axis([0 255 0 10000]); grid on;    % рисунок ліворуч
figure, imhist(f1);                % гістограма еквалізованного зображення (праворуч)
axis([0 1 0 10000]); grid on;
```





Тут видно, що рівні яскравості були зсунуті в верхню половину шкали яскравості, що призвело лише до загального освітлення зображення. Причиною такого зсуву є велика концентрація значень темних пікселів поблизу нульового рівня на початковій гістограмі. Як результат, функція перетворення, яка створюється гістограмною еквалізацією, має на початку шкали яскравості різкий стрибок і пологу основну частину.

```
hnorm = imhist(f)./numel(f);
sdf = cumsum(hnorm);
x = 0:1:255;
figure,plot(x,sdf,'k','LineWidth',4);
axis([0 255 0 1.1]); grid on;
```

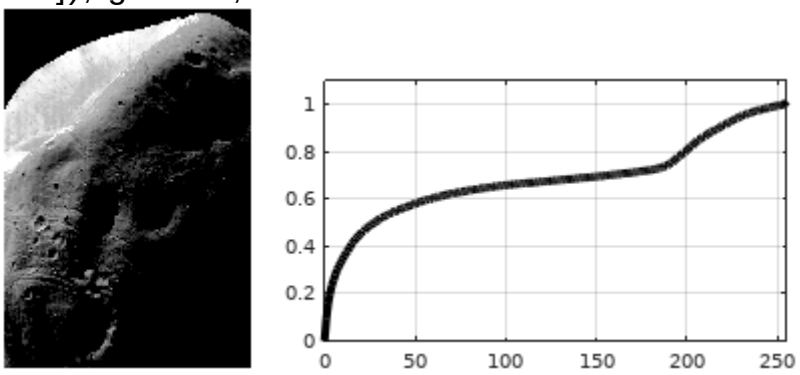


Розглядаючи останній графік, можемо дійти висновку, що треба виконати лінійне перетворення, яке покращить зображення

```
mn=min(min(f1))
mn = 0.61764
mx=max(max(f1))
mx = 0.99488
f2=(f1-mn)/(mx-mn);
figure,imshow(f2)
```

Разом еквалізація і лінійне перетворення призвели до функції перетворення зображення, графік якого показано на наступному рисунку праворуч.

```
y= (sdf-mn) /(mx-mn);
figure,plot(x,y,'k','LineWidth',4);
axis([0 255 0 1.1]); grid on;
```



Іншим можливим засобом вирішення проблеми може бути вміння задавати бажану форму гістограми результівного зображення. Метод побудови зображення із заданою гистограмою називається гістограмною підгонкою. В нашому прикладі можна спробувати створити зображення з гістограмою, яка мала б меншу концентрацію компонент в нижній частині градаційної шкали, але при цьому зберігала б загальну форму гістограми початкового зображення.

В пакеті Image Processing Toolbox (IPT) MATLAB функцію `histeq` можна застосовувати і для гістограмної підгонки, використовуючи наступний синтаксис

```
g = histeq(f,h),
```

де `f` - це вхідне зображення, `h` - задана гістограма (вектор-рядок), а `g` - вихідне зображення, гістограма якого близька до заданої гістограми `h`. Нажаль в пакеті `image` Octave ця версія функції `histeq` поки що не реалізована.

2.2. Арифметико-логічні операції з зображеннями.

При обробці зображень іноді буває корисним виконати найпростіші операції з матрицями, що їх представляють. Це ми будемо називати арифметикою зображень.

Операція складання зображень виконується функцією `imadd(...)`, яка має наступний синтаксис

```
Z = imadd(X,Y[,тип])
```

Вона додає кожний елемент масиву X до відповідного елемента масиву Y і результат розміщує у відповідний елемент Z. Масиви X і Y повинні бути числовими масивами однакового розміру і класу, або Y може бути скаляром. Результат Z має той же розмір і тип, що і X, за винятком коли X є логічним масивом, тоді Z буде типу `double`. Якщо значення елементів результату перевищують діапазон обраного типу, то вони обрізаються. Для цілих типів дробові значення округлюються. Результуюче зображення можна перетворити до більш «ємного типу» вказавши `тип` результату.

Приклад 1. *Накладання зображень.*

```
I = imread('rice.png'); imshow(I);                  % наступне зображення ліворуч
J = imread('cameraman.tif'); figure,imshow(J);  % друге зліва зображення
K = imadd(I,J); figure,imshow(K);                   % третє зліва зображення
K = imadd(I,J,'uint16'); figure,imshow(K,[ ]); % наступне зображення праворуч
```

Як бачимо, результатом додавання матриць, що представляють зображення, є накладання їх картинок.

Тут ми продемонстрували обидва варіанти використання функції `imadd`: без третього аргумента, і з його використанням для перетворення результату до типу `'uint16'`.

Нагадаємо, що один з варіантів використання функції `imshow(f,[])` для монохромних зображень означає, що використовуються аргументи

```
imshow(f,[min(f(:)) max(f(:))])
```

тобто мінімальне значення в `f` відображається чорним, а максимальне в `f` – білим.

Зауважимо, що можна виконати просте додавання матриць

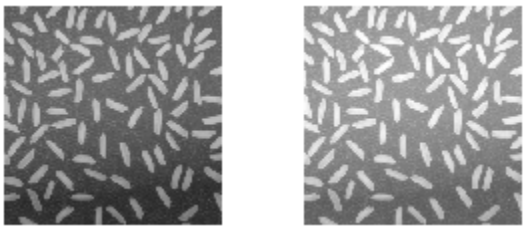
```
K=I+J; imshow(K);
```

Але тоді треба буде окремо виконувати перетворення типу.

■

Приклад 2. *Додавання сталої до зображення змінює яскравість.*

```
I = imread('rice.png');
J = imadd(I,50);
subplot(1,2,1), imshow(I)
subplot(1,2,2), imshow(J)
```



■

Для ділення зображень призначена функція `imdivide(X,Y,тип)`, де масиви X і Y повинні мати однаковий розмір і клас, або Y може бути скаляром. Використовуючи цю функцію можна, наприклад, обчислити середнє від зображень

$$C = \frac{A+B}{2}.$$

Приклад 3. *Обчислення середнього двох зображень.*

```
close all
I = imread('rice.png');
J = imread('cameraman.tif');
K = imdivide(imadd(I,J), 2);
imshow(K,[ ]); % наступне зображення ліворуч
```

або

```
K2 = imlincomb(.5,I,.5,J);
figure,imshow(K2,[ ]);  % наступне зображення праворуч
```

або

```
K3 = imdivide(imadd(I,J,'uint16'), 2);
figure,imshow(K3,[ ]);
```

Тут ми використали функцію `imdivide(X,число)`, яка виконує ділення матриці `X` зображення на `число`.

Функція `imlincomb(...)` виконує лінійне комбінування зображень. Вона має наступний синтаксис

```
Z=imlincomb(k1,img1, k2,img2,...[,тип])
```

де масиви зображень `img1,img2,...` повинні мати однаковий тип і розмір. Множники `k1,k2,...` повинні бути дійсними скалярами. Результат Z має той же розмір і тип, що і вхідні зображення, за винятком коли вони є логічними масивами, тоді Z буде типу `double`. Результуюче зображення можна перетворити до іншого типу, вказавши клас результату.

Приклад 4. *Обчислення лінійної комбінації зображень.*

```
I1 = imread('rice.png');
I2 = imread('cameraman.tif');
size(I1)
ans =   256   256
I3=imread('MoonSmall2.png');
size(I3)
ans =   100   102
A=zeros(size(I1));
```

```
A(1:100,150:251)=I3;
J1=mat2gray(I1);
J2=mat2gray(I2);
J3=mat2gray(A);
K = imlincomb(.3,J1,.7,J2,0.3,J3);
figure,imshow(K,[]);
```

Приклад 4. *Обчислення добутку зображень.*

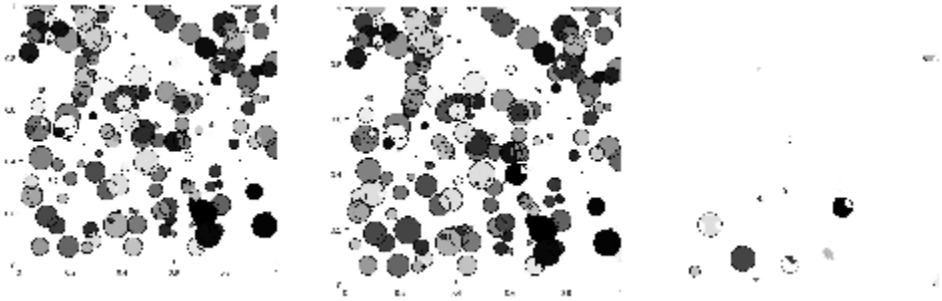
```
close all;
I=imread('Moon2.jpg');
imshow(I);
I16 = uint16(I);
J = immultiply(I16,I16);
figure, imshow(J,[ ])

K=mat2gray(I);
K2=immultiply(K,K);
K3=immultiply(K2,K);  # K3=K*K*K=K^3
figure, imshow(K3,[ ])
```

Щоб зобразити відмінність двух схожих малюнків можна використати їх різницю.

Приклад 4. *Різниця зображень.*

```
% Функция построения пузыря
% x,y - координаты центра, s - размер, c - оттенок серого цвета
rec=@(x,y,s,c) rectangle('Position',[x y s s],'Curvature',[1,1],
                                                'FaceColor',[c c c])
close all
% Рисуем 200 случайно расположенных пузырей и сохраняем в файл
arrayfun(rec,rand(1,200), rand(1,200), rand(1,200)/10, rand(1,200));
axis equal; axis([0 1 0 1]); axis off
saveas(1,'bubble1.jpg')
% Добавляем на рисунок 10 пузырей и сохраняем в файл
arrayfun(rec,rand(1,10), rand(1,10), rand(1,10)/10, rand(1,10));
```

```
saveas(1,'bubble2.jpg')
close                          % закрываем графическое окно
% Читаем изображения из файлов
f1=imread('bubble1.jpg');
f2=imread('bubble2.jpg');
g=imsubtract(f1,f2);      % Вычитаем изображения
gi=imcomplement(g);     % Строим негатив
% Отображаем рисунки
imshow(f1);
figure,imshow(f2);
figure,imshow(gi);
```



Важко помітити різницю між лівим і середнім зображеннями. Праве зображення `gi`, що представляю різницю цих малюнків, ввідображає їх відмінність.

■

Дамо деякі пояснення до коду нашого сценарію.

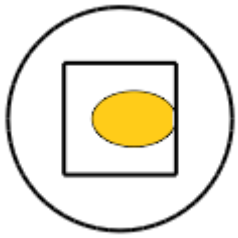
Команда **rectangle** рисует прямоугольник, эллипс или фигуру близкую им по форме. Команда `rectangle('Position',[x,y,w,h])` рисует прямоугольник с вершиной в точке (*x*,*y*) шириной *w* высотой *h*. Свойство **Curvature** определяет кривизны сторон.

`rectangle('Position',[x,y,w,h], 'Curvature',[`c_g`,`c_v`])`

Если оно опущено или задано равным [0 0], то отрезки сторон будут прямые, и мы получаем прямоугольник. Задание значений кривизны в интервале от 0 до 1 искривляет стороны фигуры – от прямолинейных (при c_g=0 или c_v=0) до эллиптических (при c_g=1 или c_v=1). Первое число в паре [c_g,c_v] управляет кривизной горизонтальных сторон, второе – вертикальных. Слово кривизна используется только для обозначения параметра функции и не является кривизной в точном математическом смысле этого значения. Можно использовать одно число, тогда оно относится к меньшей из сторон фигуры. Наприклад

```
clf                % очищення активного вікна або створення порожнього
rectangle('Position',[-2 -2 4 4], 'Curvature',[1,1],'LineWidth',2);
                                                    % зовнішній круг
axis equal;
% додавання малюнків нових фігур
rectangle('Position',[-1 -1 2 2], 'LineWidth',2);        % квадрат
rectangle('Position',[-0.5 -0.5 1.5 1], 'Curvature',[1,1],
                          'FaceColor', [1 0.8 0.1]);  % ззалитий еліпс
```

Побудовані цими інструкціями фігури зображені на наступному рисунку.



Опция `'FaceColor',[r g b]` приводит к построению закрашенной фигуры цветом формате RGB , где $0 \le r, g, b \le 1$.

Функція

```
rec=@(x,y,s,c) rectangle('Position',[x y s s], 'Curvature',[1,1],
                                                'FaceColor', [c c c])
```

малює коло з центром в точці `x,y`, діаметром `s`, яке зафарбовано сірим кольором зі значенням `c`.

Функція `arrayfun(...)` в форматі

```
Z=arrayfun(func,X,Y,...)
```

виконує функцію `func(x,y,...)` на кожному наборі елементів з масивів `X,Y,...`. Наприклад, якщо ви маєте функцію двох аргументів

```
f=@(x,y) x^2+y^2
```

то допустима наступна інструкція

```
arrayfun (f, [3, 5], [4, 2])
ans =   25   29
```

В нашому прикладі функцію

```
arrayfun(rec,rand(1,200),rand(1,200),rand(1,200)/10,rand(1,200));
```

ми застосували для визову функції `rec`, передавши їй 200 наборів аргументів, які зібрані з випадкових чисел.

Для збереження малюнка, який побудовано в графічному вікні, ми використали функцію `saveas(дескриптор вікна, ім'я файла)`. Якщо дескриптор опущено, то зберігається вміст активного вікна.

```
axis off      # настроювання графічного вікна
h=gcf();      # отримання дескриптора активного вікна
saveas(h,'figure1.png')  # зберігання малюнка в файл
```

Можливо також зберігати зображення в графічний файл, використовуючи функцію `print`. Наприклад,

```
print -djpg figure1.jpg
print -dpng figure2.png
```

Тут використана позначка `-d` (device) і зразу без проміжку вказано формат графічного файла. Потім вказано ім'я файла.

Додаток.

А. Код функції побудови ламаної по координатах вузлів.

```
function y=brokenline(x,pList)
% обчислення y(x) по рівнянню ламаної
% pList =[x1,y1,x2,y2,...,xn,yn] містить пари координат
% вузлів ламаної
% перевірки хибного визову не виконуються
% вузлів повинно бути не меньше двох,
% координати x монтонно зростають
  n=length(pList)/2;
  x0=pList(1); y0=pList(2);
  x1=pList(3); y1=pList(4);
  xn=pList(2*n-1); yn=pList(2*n);
  xnp=pList(2*n-3); ynp=pList(2*n-2);

  s=(y0+(y1-y0)/(x1-x0).*(x-x0)+ynp +
                          (yn-ynp)/(xn-xnp).*(x-xnp))/2;
  for i=2:n-1
    xk=pList(2*i-1);
    yk=pList(2*i);
    xkp=pList(2*i-3);
    ykp=pList(2*i-2);
    xkn=pList(2*i+1);
    ykn=pList(2*i+2);
    s=s+((ykn-yk)/(xkn-xk)-(yk-ykp)/(xk-xkp)).*abs(x-xk)/2;
  endfor
  y=s;
```

Б. Код функції побудови сплайна-сходинки по координатах вузлів. Обчислення виконуються по наступній рекурентній формулі.

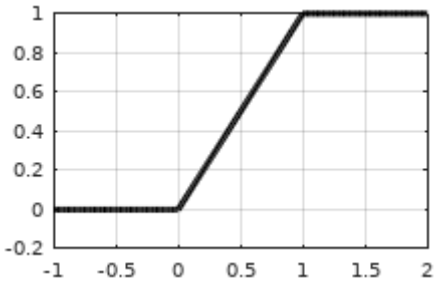
$$P_n(x, x_0, x_1, \ldots, x_{n-1}, x_n) = \\ = \frac{(x - x_0)P_{n-1}(x, x_0, \ldots, x_{n-1}) + (x_n - x)P_{n-1}(x, x_1, \ldots, x_n)}{x_n - x_0}$$

```
% сплайн-сходинка, значення змінюється від 0 до 1
% при pnt=[x0 x1] маємо ламану з вузлами точках x0 і x1
% більша кількість вузлів приводить до гладкого сплайну
function y=stepspline(x,pnt)
  % повинно бути n=2,3,4,...
  n=length(pnt);
  x0=pnt(1);
  x1=pnt(2);
  xn=pnt(n);
  if (n==2)
    y=(x1-x0+abs(x-x0)-abs(x-x1))/2/(x1-x0);
  else
```

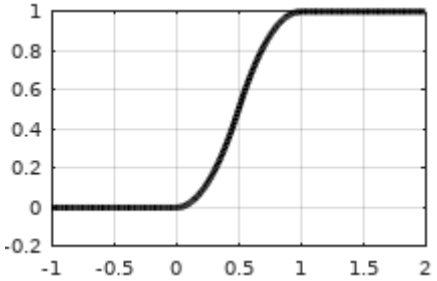
```
    pnt0=pnt(1:n-1);
    pntn=pnt(2:n);
    y=((x-x0).*stepspline(x,pnt0)+(xn-x).*
                                stepspline(x,pntn))/(xn-x0);
  endif
```

Приклади використання.

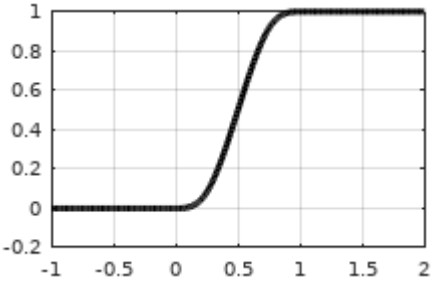
```
x=-1:0.01:2; y=stepspline(x,[0 1]);
plot(x,y,'k','LineWidth',3); grid on;
```



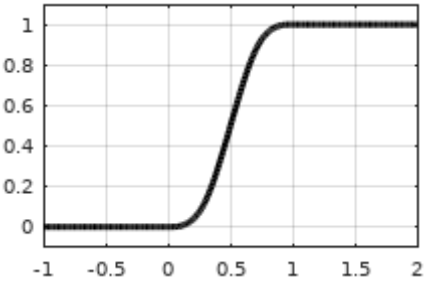

```
x=-1:0.01:2; y=stepspline(x,[0 0.5 1]);
plot(x,y,'k','LineWidth',3); grid on;
```




```
x=-1:0.01:2; y=stepspline(x,[0 0.25 0.75 1]);
plot(x,y,'k','LineWidth',3); grid on;
```




```
x=-1:0.01:2; y=stepspline(x,[0 0.3 0.7 1]);
plot(x,y,'k','LineWidth',3); grid on;
axis([-1 2 -0.1 1.1]);
```



Дробная часть числа.

```
frPart=@(x) x-fix(x);
```