



## Математические методы обработки изображений.

В настоящем пособии излагаются методы, которые используются при обработке изображений. Перед вами часть, которая знакомит с методами обработки цветных цифровых изображений.

### Методы обработки цветных цифровых изображений.

#### Оглавление

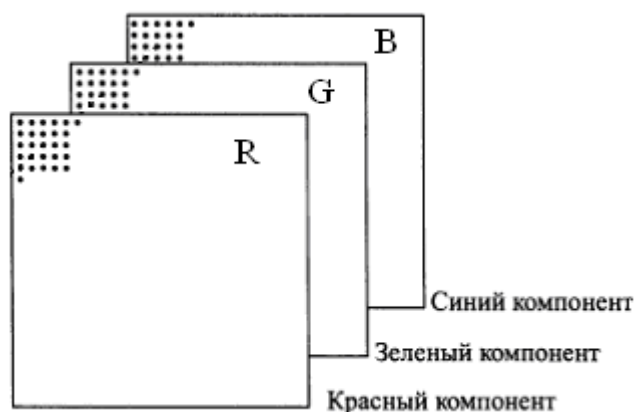
|  |    |
|--|----|
| 4. Обработка цветных изображений. ....               | 1  |
| 4.1 Представление цветных изображений в MATLAB ..... | 1  |
| 4.2 Основы обработки цветных изображений .....       | 14 |
| Литература .....                                     | 25 |

## 4. Обработка цветных изображений.

### 4.1 Представление цветных изображений в MATLAB

В пакете IPT цветные изображения представимы в двух видах: в виде RGB (Red, Green, Blue) изображений и в виде индексированных изображений.

Цветным RGB изображением называется массив  $M \times N \times 3$ , состоящий из трех матриц размера  $M \times N$ , которые соответствуют трем цветовым компонентам: красному, зеленому и синему. Расположение монохромных пикселей, соответствующих этим матрицам на цветном RGB изображении, показано на следующем рисунке.



RGB изображение можно представлять себе в виде «пакета» трех монохромных изображений с градацией серого цвета, которые подаются на красный, зеленый и синий входы цветного монитора, в результате чего формируется цветное изображение на экране. Три монохромных изображения, формирующих единое RGB изображение, принято называть красной, зеленой и синей составляющей (компонентой) изображения. Класс компонент изображения определяет область их значений. Если класс данных цветовой компоненты есть `double`, то область значений — интервал  $[0,1]$ . Аналогично, области значений  $[0,255]$  или  $[0,65535]$ , соответственно, у RGB изображений класса `uint8` и `uint16`. Число битов, используемых для представления величины цветного пиксела по всем составляющим RGB, называется глубиной цвета изображения. Например, если каждая компонента является 8-ми битовым изображением, то соответствующее RGB изображение имеет глубину 24 бита. Обычно, составляющие имеют одинаковое число битов. При 8-ми битном представлении каждой компоненты всего имеется  $16777216$  цветов.

Пусть `fR`, `fG` и `fB` обозначают три составляющие RGB изображения. Соответствующее им цветное GRB изображение строится с помощью оператора `cat` (concatenation – соединение, сцепление):

```
rgbimage = cat(3, fR, fG, fB)
```

Порядок расположения цветовых составляющих в этой формуле весьма важен. В общем случае, оператор `cat(dim, A1, A2, ...)` связывает массивы вдоль размерности, обозначенной в переменной `dim`. Например, если `dim = 1`, то массивы располагаются по вертикали, если `dim = 2`, то — по горизонтали, а если `dim = 3`, то они укладываются по третьему измерению так, как показано на предыдущем рисунке. Т.е. для конструирования RGB изображения из трех компонент мы должны полагать `dim=3`.

Если все три составляющие изображения идентичны, то в результате получается изображение серого цвета.

**Пример.** Извлечь цветовые составляющие RGB.

```
f=imread('Flowers.tif');imshow(f);% исходное изображение (слева)
size(f)
ans = 600 600 3
imshow(f(:,:,1)); % красная составляющая (2-я слева)
imshow(f(:,:,2)); % зеленая составляющая (3-я слева)
imshow(f(:,:,3)); % синяя составляющая (справа)
```



Вы можете обработать любую составляющую как монохромное изображение, а затем с помощью функции `cat` соединить их в цветное изображение.

**Пример.** Построить негатив к RGB изображению класса `uint8`.

```
f=imread('Chalk.tif'); imshow(f); % рисунок слева
g1=255-f(:,:,1); g2=255-f(:,:,2); g3=255-f(:,:,3);
gn=cat(3,g1,g2,g3); imshow(gn); % рисунок справа
```



Вычитание из 255 следует из того, что исходное изображение относится к классу `uint8`.

Конечно, то же негативное цветное изображение можно получить одной командой

```
g = imcomplement(f); imshow(g);
```

Аналогично можно построить «усредненное» монохромное изображение

```
f=imread('Flowers.tif');
fd1=double(f(:,:,1)); imshow(fd1/255); %образ r цветовой плоскости
fd2=double(f(:,:,2)); imshow(fd2/255); %образ g цветовой плоскости
fd3=double(f(:,:,3)); imshow(fd3/255); %образ b цветовой плоскости
fdm=fd1+fd2+fd3; % суммируем цветные карты
max(max(fdm))
ans =
    765
fd=fdm/max(max(fdm));
imshow(fd); % усредненное черно-белое изображение
```

Однако для вычисления яркости изображения лучше использовать относительные коэффициенты чувствительности человеческого глаза к RGB цветам. В этом случае монохромная яркость в соответствии с NTSC стандартом вычисляется по формулу

```
I=0.299*rgbimg(:,:,1)+0.587*rgbimg(:,:,2)+0.114*rgbimg(:,:,3);
```

Заметим, что сумма коэффициентов равна единице. Тогда для получения монохромного изображения из цветного 'football.jpg' можно выполнить команды

```
f=imread('football.jpg');
I=0.299*f(:,:,1)+0.587*f(:,:,2)+0.114*f(:,:,3);
imshow(I);
```

Кроме RGB модели представления изображений существуют и другие модели, более подходящие для технических целей или ближе приспособленные к человеческому восприятию.

Цветовая модель  $YCbCr$  широко используется в цифровом видео. В этой модели компонента  $Y$  называется «светлота» (яркость), а цвет хранится в

виде двух разностных цветовых компонент  $C_b$  и  $C_r$ . Величина  $C_b=B-Y$  — это разность между голубой компонентой  $B$  и светлотой  $Y$ , а  $C_r=R-Y$  — это разность между красной компонентой  $R$  и  $Y$ . Светлота или светлота цветного изображения соответствует характеристике интенсивность (полутоновая яркость) в монохромном случае. Обычно мы употребляем термин «яркость». Она (яркость – уровень серого цвета) является основной характеристикой монохромных (полутоновых) изображений.

При конвертации из формата RGB в  $YC_bC_r$  используются уравнения

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.000 \\ 112.000 & -93.786 & -18.214 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Функция преобразования MatLab имеет вид

`YCbCr_image=rgb2ycbcr(rgb_image)`

Обратное преобразование выполняется функцией

`rgb_image=ycbcr2rgb(YCbCr_image)`

```
f=imread('Flowers.tif');imshow(f);
z=rgb2ycbcr(f);
imshow(z);
imshow(z(:,:,1));
imshow(z(:,:,2));
imshow(z(:,:,3));
```



Слева направо: светлота (компонента  $Y$ );  $C_b=B-Y$ ;  $C_r=R-Y$ ; ( $B$  – синяя компонента,  $R$  - красная компонента).

Цветовая система NTSC используется в телевидении. Основное преимущество данного формата заключается в том, что в нем информация о яркости (насыщенности), эквивалентная шкале серого цвета в черно-белом телевидении, отделена от самих цветовых данных, что позволяет использовать один и тот же сигнал в цветных и черно-белых телевизионных приемниках. В формате NTSC пиксел представлен тремя компонентами: светлота ( $Y$ ), цветовой тон ( $I$ ) и насыщенность ( $Q$ ), где выбор букв  $YIQ$  является общепринятым. Светлота содержит информацию о яркости в шкале градаций серого, а две другие компоненты несут информацию о цвете в телевизионном сигнале. Компоненты  $YIQ$  получаются из тройки RGB с помощью следующего преобразования:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Отметим, что сумма элементов первой строки матрицы равна 1, а сумма элементов по остальным двум строкам равна 0. Это легко понять, так как яркости всех трех компонент RGB монохромного изображения должны быть одинаковы, а значит, для такого изображения компоненты I и Q равны нулю (если справа R=G=B, то слева должно получиться I=0 и Q=0).

Функция `rgb2ntsc` совершает это преобразование:

```

yiq_image = rgb2ntsc(rgb_image),

```

где входное RGB изображение может быть класса `uint8`, `uint16` или `double`. Выходом служит массив  $M \times N \times 3$  класса `double`. Компонента светлота получается командой `yiq_image(:, :, 1)`, цветовой тон — `yiq_image(:, :, 2)`, а насыщенность — `yiq_image(:, :, 3)`.

```

f=imread('Flowers.tif');imshow(f);
z=rgb2ntsc(f);
imshow(z(:, :, 1)); % компонента 'светлота'

```



Фактически, как мы видели ранее, компонента «светлота» создает монохромное представление исходного цветного изображения.

Вычислим матрицу обратного преобразования

```

A=[0.299 0.587 0.114; 0.596 -0.274 -0.322; 0.211 -0.523 0.312]

```

```

A =
    0.2990    0.5870    0.1140
    0.5960   -0.2740   -0.3220
    0.2110   -0.5230    0.3120

```

```

B=inv(A)
B =
    1.0000    0.9562    0.6214
    1.0000   -0.2727   -0.6468
    1.0000   -1.1037    1.7006

```

Это значит, что компоненты RGB получаются из YIQ с помощью обратного преобразования

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0000 & 0.9562 & 0.6214 \\ 1.0000 & -0.2727 & -0.6468 \\ 1.0000 & -1.1037 & 1.7006 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

которое в IPT реализовано в виде функции

```

rgb_image=ntsc2rgb(yiq_image);

```

Здесь и входное, и выходное изображения имеют класс `double`.

Цветовые пространства CMY и CMYK. Голубой, пурпурный и желтый являются вторичными основными цветами световых источников. Большинство аппаратов для нанесения цветных красителей на бумагу, вроде цветных принтеров и устройств копирования в цвете, либо требуют

представления цветных данных в формате CMY, либо автоматически совершают преобразование из RGB в CMY. Это преобразование совершается по очень простым правилам

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

где предполагается, что все цветовые компоненты нормированы в диапазоне [0,1]. Эти уравнения демонстрируют эффект неотражения красного цвета от поверхности, покрытой чисто голубой краской (из уравнений имеем  $C=1-R$ ). Аналогично, чисто пурпурный цвет не отражает зеленый, а чисто желтый не отражает синий. Из этих уравнений следует, что для получения RGB компонент из CMY достаточно вычесть каждую из них из 1.

В теории равные количества первичных красителей голубого, пурпурного и желтого должны давать черный цвет. На практике, при смешении этих цветов на печати получается черный, который выглядит более осветленным, чем оригинальный черный цвет. Поэтому, чтобы получить истинный черный цвет ((который часто доминирует при цветной печати), цветовая модель CMY расширяется до модели CMYK четвертым цветовым компонентом — черным цветом. Таким образом, когда издатели говорят о четырехцветной печати, они имеют в виду трехцветную модель CMY плюс черный цвет.

Функцию `imcomplement`, (взятия негатива) можно использовать для преобразования из RGB в формат CMY:

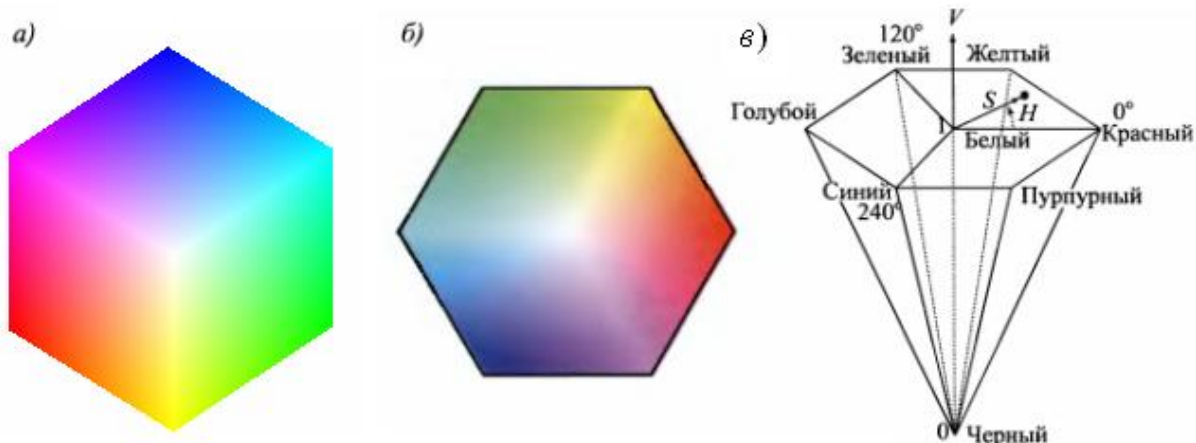
```
cmu_image = imcomplement (rgbimage).
```

Эту же функцию можно использовать для обратного преобразования:

```
rgb_image = imcomplement (cmuimage).
```

Цветовая система HSV (Hue, Saturation, Value: цветовой тон, насыщенность, величина) представляет собой одно из цветовых пространств, в которых выбор цвета похож на обращение с палитрой или гаммой цветов при работе художника с красками или чернилами. Пользуясь терминологией художников, термины цветовой тон, насыщенность и величина соответствуют приблизительно таким словам, как окраска, оттенок и сила тона.

Цветовое пространство HSV можно увидеть, если посмотреть на цветовой куб RGB (рис. а) вдоль его серой оси, которая соединяет черную и белую вершины. Цветовой куб получается, если из одной из вершин куба в направлениях ребер менять «количество» красной, зеленой и синей составляющей. В результате наблюдается цветовая палитра в форме шестиугольника, которая изображена на следующем рисунке б).



а) цветовой куб; б) цветовой шестиугольник; в) шестигранный конус HSV

По мере движения вдоль серой оси куба от его черной вершины к белой принято считать, что  $V$  изменяется от 0 до 1. Это принято изображать в виде шестиугольной пирамиды (рисунок в), причем размер шестиугольника в перпендикулярном сечении соответствует величине  $V$  (см. рис. в). Величина ( $V$ ) измеряется вдоль оси конуса и черная вершина соответствует величине  $V = 0$ . Цветовой тон  $H$  выражается углом на цветовом шестиугольнике, причем принято отсчитывать этот угол от луча, соединяющего центр и красную вершину. Насыщенность (чистота)  $S$  цвета измеряется расстоянием до оси  $V$ . Таким образом, на оси конуса располагаются все оттенки серого цвета.

Функция MATLAB для преобразования из RGB в HSV имеет вид

```
hsv_image=rgb2hsv(rgb_image);
```

Входное RGB изображение может иметь класс uint8, uint16 или double, а выходное имеет класс double. Обратное преобразование из HSV в RGB совершается функцией

```
rgb_image=hsv2rgb(hsv_image);
```

Входное изображение должно принадлежать классу double, а выходное имеет класс double.

Вот пример HSV составляющих

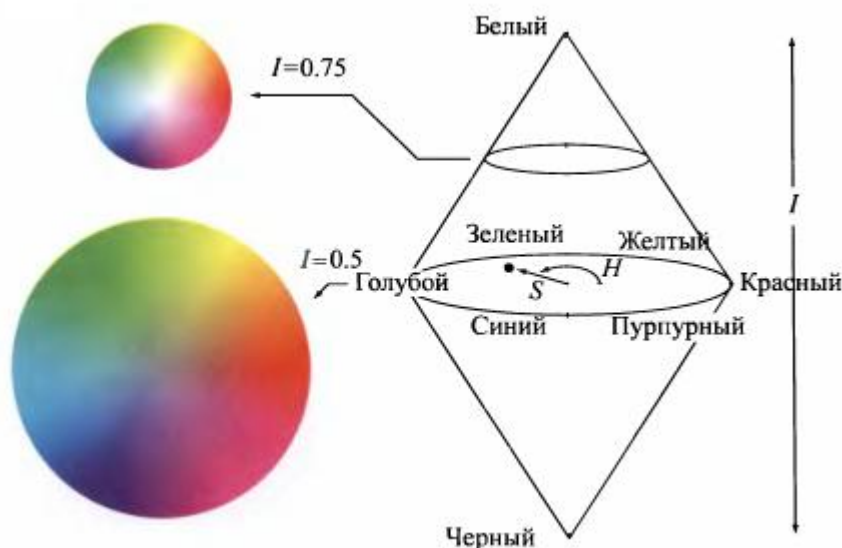
```
f=imread('Flowers.tif');imshow(f);
hsv = rgb2hsv(f);          % преобразование в модель HSV
imshow(hsv(:,:,1));       % цветовой тон (слева)
imshow(hsv(:,:,2));       % насыщенность (в центре)
imshow(hsv(:,:,3));       % величина (справа)
```



Если не брать HSV, другие рассмотренные выше цветовые модели плохо приспособлены к описанию цветов, свойственному для человека. Например, говоря о цвете автомобиля, мы никогда не скажем о процентном соотношении в нем красного, зеленого и синего. Описывая объект, мы скажем о его цвете (цветовом тоне), насыщенности и яркости (светлоте). Цветовой тон сообщает собственно цвет и его чистоту (чисто красный, оранжевый, желтый и т.п.), насыщенность дает меру того, насколько чистый цвет разбавлен белым. Светлота в монохромном случае соответствует термину «яркость»

В цветовом пространстве HSI (Hue, Saturation, Intensity: цветовой тон, насыщенность, интенсивность), также как и в HSV, яркостная информация (интенсивность) отделена от цветовой информации (цветовой тон и насыщенность) цветного изображения. В результате модель HSI является важным инструментом в алгоритмах обработки цветных изображений, поскольку в ее основе лежит описание цвета, интуитивно понятное человеку. В этом смысле цветовое пространство HSV похоже на HSI, но HSV сфокусировано на представлении цветов в виде палитры художника, поэтому оно менее удобно в цифровых приложениях.

На следующем рисунке представлена модели HSI на основе цветового круга.



Цветовой круг расположен перпендикулярно оси интенсивности. Цветовой тон  $H$  измеряется углом, отсчитываемым от направления на красную точку круга, а насыщенность  $S$  соответствует относительному расстоянию текущей точки до центра круга.

Индексированные изображения. Индексированное изображение имеет две компоненты: числовую матрицу данных  $X$  и матрицу цветовой карты  $map$ . Матрица  $map$  представляет собой массив размеров  $m \times 3$  класса `double`, в котором записаны вещественные числа с плавающей запятой из интервала  $[0,1]$ . Длина  $m$  цветовой карты равна числу различных цветов на данном изображении. Элементы строки матрицы  $map$  обозначают



«количество» красной, зеленой и синей компоненты цвета. Цвет каждого пиксела определяется с помощью числа, стоящего в матрице  $X$ , которое используется как указатель на строку  $map$ . Если  $X$  имеет класс `double`, то все его компоненты с величинами, меньшими или равными 1, указывают на первую строку  $map$ , все компоненты с величинами между 1 и 2 включительно указывают на вторую строку и т.д. Если  $X$  принадлежит классу `uint8` или `uint16`, то все его компоненты, равные 0, указывают на первую строку  $map$ , все компоненты, равные 1, указывают на вторую строку  $map$  и т. д. Этот метод определения цвета пикселей проиллюстрирован на следующем рисунке.



Для отображения на дисплее индексированного изображения следует выполнить команду

```
imshow(X, map);
```

или, что эквивалентно,

```
image(X);
```

```
colormap(map);
```

В графическом файле цветовая карта хранится вместе с индексированным изображением, и она автоматически загружается, когда функция `imread` считывает изображение. Например, следующая команда загружает в матрицу  $A$  индексированное изображение, а карту цветов в матрицу  $B$ .

```
[A,B]=imread('Pogorelov32x32.bmp');
```

Чтобы правильно представить изображение надо выполнить команду

```
imshow(A,B);
```

Имеется несколько способов задания цветовой карты. Первый способ состоит в задании матрицы  $map$  в виде

```
map(k, :) = [r(k) g(k) b(k)];
```

где  $[r(k) \ g(k) \ b(k)]$  — это RGB значения, которые определяют строку карты с номером  $k$ . Карта будет заполняться при изменении параметра  $k$ .

Например, команда

```
map=[0 0 0; 0.25 0.25 0.25; 0.5 0.5 0.5; 0.75 0.75 0.75; 1 1 1];
```

создаст карту серых цветов, состоящую из пяти оттенков.

Для использования основных цветов можно использовать три формата их задания. Длинное или короткое имя, заключенное в кавычки, может использоваться в командах наравне с числовым триплетом.

RGB значения некоторых основных цветов.

| Длинное имя | Короткое имя | RGB значения |
|-------------|--------------|--------------|
| Black       | k            | [0 0 0]      |
| Blue        | b            | [0 0 1]      |
| Green       | g            | [0 1 0]      |
| Cyan        | c            | [0 1 1]      |
| Red         | r            | [1 0 0]      |
| Magenta     | m            | [1 0 1]      |
| Yellow      | y            | [1 1 0]      |
| White       | w            | [1 1 1]      |

Например, цвет фона окна можно поменять на зеленый любой из следующих трех команд:

```
whitebg('g') ;
whitebg('green') ;
whitebg([0 1 0]);
```

Для обозначения цветов, не представленных в таблице, используются дробные числа. Например, [.5 .5 .5] — серый цвет, [.5 0 0] — темно-красный, а [.49 1 .83] — аквамаринный.

В MATLAB имеется несколько стандартных цветовых карт, которые можно загрузить командой

```
colormap(map_name);
```

В результате матрица цветовой карты становится равной map\_name. Например,

```
colormap(copper);
```

где copper — одна из цветовых карт, имеющихся в MATLAB. Цвета в этой карте гладко меняются от черного до светло-бронзового. Если последнее изображение, выведенное на экран, было индексированным, то после этой команды цветовая карта поменяется на copper. Вот имена цветовых карт MATLAB: autumn, bone, colorcube, cool, gray, hot, hsv, jet, lines, pink, prism, spring, summer, white, winter. Длины (число цветов) этих карт можно ограничить, поставив в круглых скобках соответствующее число. Например, gray(16) задает цветовую карту из 16 оттенков серого цвета.

Иногда бывает необходимо аппроксимировать индексированное изображение другим изображением с меньшим набором цветов. Для этой цели используется функция imapprox, вызов которой имеет вид

```
[Y, newmap] = imapprox(X, map, n);
```

Эта функция возвращает массив Y с цветовой картой newmap, который имеет не более n цветов. Входной массив может принадлежать классу uint8, uint16 или double. Выход Y принадлежит классу uint8, если n не превосходит 256. Если n больше 256, то Y принадлежит классу double.

Когда число строк матрицы map меньше числа различных значений X, «лишние» значения в X отображаются с помощью одного и того же цвета в

map. Например, пусть X состоит из четырех вертикальных полос одинаковой ширины со значениями 1, 64, 128 и 256. Если мы определим цветовую карту map = [0 0 0; 1 1 1], то все элементы X со значениями 1 будут указывать на первую строку map ((черный цвет), а все остальные значения X будут указывать на вторую строку map (белый цвет). Таким образом, команда imshow(X, map) покажет изображение, на котором за черной полосой следуют три белые полосы. Это так и будет до тех пор, пока длина карты не станет равна 65, в этот момент на дисплее будет отображена черная полоска, затем серая полоска, за которыми следуют две белые. В результате подобных действий на экране может появиться весьма странное изображение, если длина карты превосходит допустимый диапазон значений элементов X.

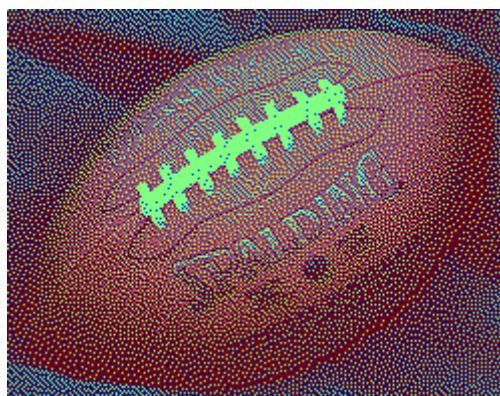
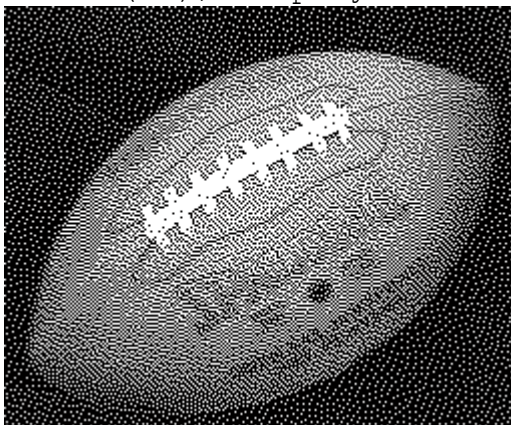
Рассмотрим некоторые функции IPT для обращения с RGB и индексированными изображениями.

Функцию dither (дрожать) можно применять как к монохромным, так и к цветным изображениям. Процесс дрожания используется в печатном деле для имитации градации серого цвета при печати черными точками. Для монохромных изображений функция dither пытается передать градации серого цвета с помощью черных точек на белом фоне. Плотность черных точек уменьшается на ярких областях и увеличивается на более темных областях. Алгоритм, применяемый при дрожании, основан на компромиссе между «точностью» зрительного восприятия и вычислительной сложностью. Синтаксис функции dither при работе с монохромными изображениями имеет вид

```
bw = dither(gi);
```

где gi — это монохромное изображение, а bw — результат дрожания (двоичное изображение).

```
f=imread('football.jpg');
f1=f(:,:,1); % Выделение одной цветовой плоскости
C1=dither(f1);
imshow(C1); % рисунок слева
```



Обратите внимание на тип матрицы C1 – logical.

Исходное изображение может быть цветным. Тогда функция dither создает «дрожанием» индексированное цветное изображение,

приближающее исходное RGB изображение. В этом случае при создании индексированного изображения вторым аргументом в `dither` надо передать палитру и потом отображать полученное изображение в этой палитре

```
f=imread('football.jpg');
C=dither(f, jet); imshow(C, jet); % рисунок справа
```

или с другими палитрами

```
C=dither(f, jet(16)); imshow(C, jet(16));
C=dither(f, colorcube); imshow(C, colorcube);
C=dither(f, hot); imshow(C, hot);
C=dither(f, autumn(2)); imshow(C, autumn(2));
```

Функция `rgb2gray` преобразует RGB изображение в монохромное. Фактически она отбрасывает тона и насыщенности, но оставляя яркость.

```
f=imread('Flowers.tif'); imshow(f); % рисунок слева
g=rgb2gray(f); imshow(g); % рисунок справа
```



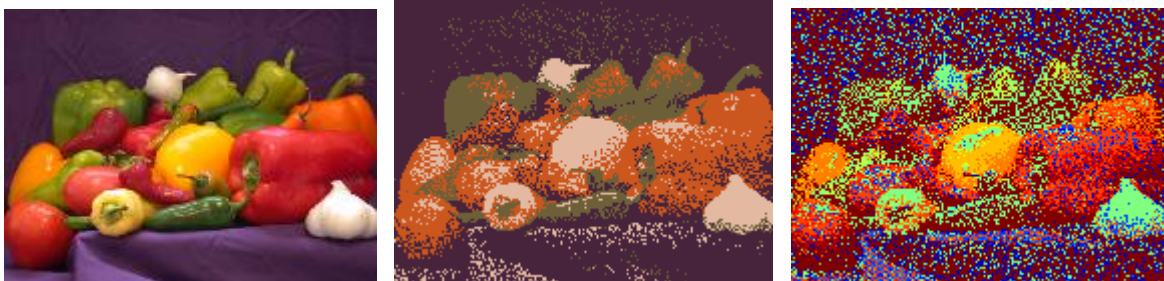
Напомним, что для того, чтобы определить тип изображения можно выполнить команду

```
imfinfo('Flowers.tif')
```

которая выводит много различной информации о графическом файле и, в частности, его тип.

Функция `rgb2ind` строит индексированное изображение из RGB изображения. Вторым ее аргументом может быть количество цветов в палитре или новая палитра индексированного изображения

```
f = imread('peppers.png'); imshow(f); % рис. слева
[g, map] = rgb2ind(f, 4); imshow(g, map); % рис. в центре
[g, map] = rgb2ind(f, jet(16)); imshow(g, map); % рис. справа
```



Есть еще один вариант вызова этой функции

```
[X, map] = rgb2ind(gray_image, n, dither_option);
```

где `n` определяет длину (или число цветов) карты `map`, а символьный параметр `dither_option` может иметь одно из двух значений: `'dither'` (по умолчанию), если необходимо получить лучшее цветовое разрешение за

счет пространственного; и, наоборот, 'nodither', тогда каждый цвет исходного изображения отображается в ближайший цвет новой карты (зависящий от величины n). В этом случае дрожание не делается.

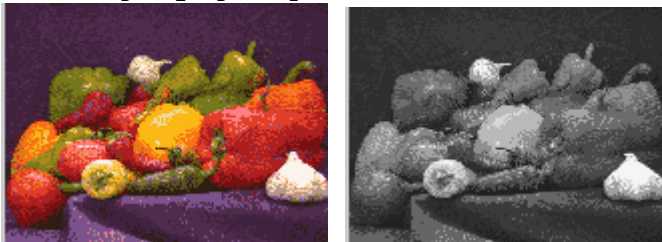
```
f=imread('Flowers.tif'); imshow(f); % фото слева
[X1, map1] = rgb2ind(f, 8, 'nodither');
imshow(X1, map1); % фото в середине
[X2, map2] = rgb2ind(f, 8, 'dither');
imshow(X2, map2); % фото справа
```



Оба (среднее и правое) изображения имеют всего 8 цветов, т. е. было совершено значительное сокращение числа возможных цветов в  $f$ , которое для 24-битового RGB изображения, как уже отмечалось, превосходит 16 миллионов цветов. На среднем рисунке заметны ложные контуры, особенно в центре большого цветка. Изображение, построенное с дрожанием, лучше передает цветовые оттенки; на нем имеется меньше ложных контуров. В этом проявляется эффект «хаотичности», который вносится при дрожании. Изображение выглядит слегка размытым, однако для глаза оно будет казаться более точным.

Функция `ind2gray` строит монохромное изображение из индексированного. Аргументами этой функции должны быть изображение и его палитра.

```
f = imread('peppers.png');
[g, map]=rgb2ind(f,16); %g-индексированное изображение
imshow(g, map);
h=ind2gray(g, map); imshow(h); % h - монохромное изображение
```



Функция `gray2ind`, имеет синтаксис

```
[X, map] = gray2ind(grayimage, n);
```

масштабирует, а затем округляет изображение `grayimage` для получения индексированного изображения `X` с цветовой картой `gray(n)`. Если параметр `n` опущен, то по умолчанию `n = 64`. Входное изображение может быть класса `uint8`, `uint16` или `double`. Выходное изображение `X` принадлежит классу `uint8`, если `n` меньше 256, а если `n` больше этого

числа, то оно имеет класс `uint16`. Например (изображение `h` из предыдущего примера)

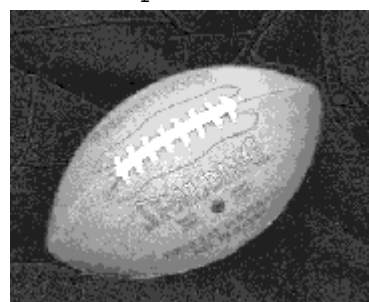
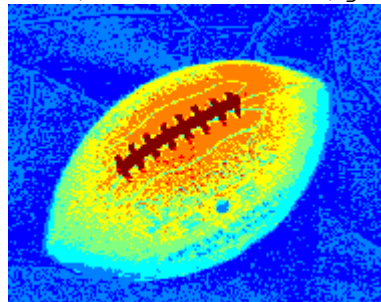
```
[X, map] = gray2ind(h);  
imshow(X, map);  
% индексированное изображение можно отобразить в другой палитре  
imshow(X, summer);
```

Функция `grayslice` имеет синтаксис

```
X = grayslice(grayimage, n);
```

Она строит индексированное изображение с уменьшенным числом цветов.

```
f=imread('football.jpg');  
f1=f(:,:,1);  
D1=grayslice(f1,8);imshow(D1,jet(8)); % рис. слева
```



Как уже отмечалось, полученное индексированное изображение можно увидеть на экране после выполнения команды `imshow(X, map)` с использованием карты подходящей длины (например, `jet(8)`).

Другая форма этой команды выглядит так:

```
X = grayslice(grayimage, v),
```

где `v` — это вектор, компоненты которого используются в качестве пороговых значений при разделении цветов в `grayimage`. Например

```
v=[16 32 48 64 96 128 160 192 256];
```

```
X=grayslice(f1,v);
```

```
max(max(X))
```

```
ans =
```

```
9
```

```
imshow(X, gray(9)); % рис. выше справа
```

В сочетании с цветовыми картами, функция `grayslice` является основным инструментом при обработке псевдоцветных изображений, когда специальная шкала оттенков серого цвета используется для обозначения цветов.

## 4.2 Основы обработки цветных изображений

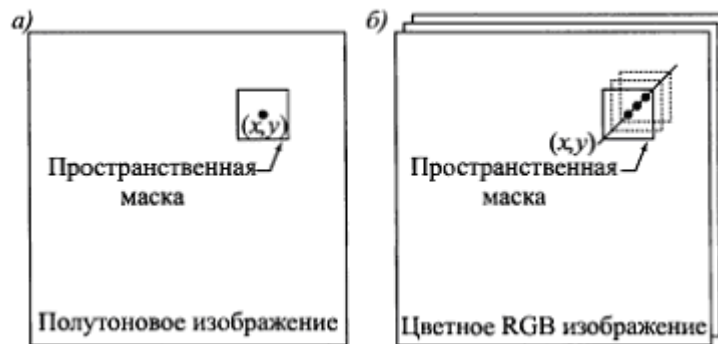
Разделим подходы, используемые при обработке цветных изображений, на три категории: преобразование цветов; отдельная пространственная обработка цветовых плоскостей; обработка цветовых векторов. Процедуры первой категории обрабатывают пиксели каждой цветовой плоскости, основываясь исключительно на значениях пикселей, без учета их пространственных координат. Эти подходы аналогичны методам,

рассмотренным при изучении преобразований яркости монохромных изображений. Методы второй категории основаны на пространственной фильтрации отдельных цветовых плоскостей, и они аналогичны пространственной фильтрации монохромных изображений. К третьей категории относятся методы, которые совершают обработку компонент цветного изображения как единого целого.

Поскольку цветные изображения имеют, по крайней мере, три компоненты, цветные пиксели можно рассматривать как трехмерные вектора. Например, в системе RGB каждую цветовую точку можно рассматривать как вектор, проведенный из начала координат в соответствующую точку цветового пространства. Если  $c$  — это произвольный вектор в цветовом пространстве RGB, то

$$c(x, y) = \begin{bmatrix} c_R(x, y) \\ c_G(x, y) \\ c_B(x, y) \end{bmatrix} = \begin{bmatrix} R(x, y) \\ G(x, y) \\ B(x, y) \end{bmatrix}$$

В качестве иллюстрации, на следующем рисунке показана пространственная окрестностная обработка монохромного и цветного изображения.



Пространственные маски для монохромного и цветного RGB изображений.

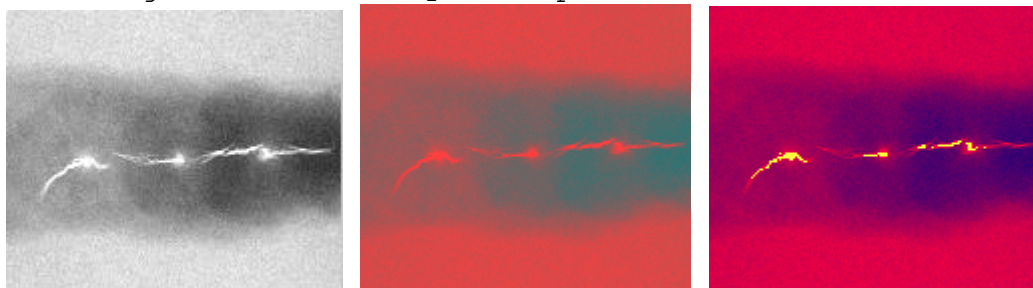
Предположим, что процедура заключается в окрестностном осреднении. Она линейная. На рисунке (a) осреднение делается суммированием яркости всех полутоновых пикселей окрестности и делением полученной суммы на общее число пикселей окрестности, а на рисунке (б) следует просуммировать все векторы окрестности и разделить каждую компоненту полученного суммарного вектора на общее число векторов окрестности. Каждая компонента осредненного вектора равна сумме значений, соответствующих данной компоненте, деленной на число пикселей окрестности, что совпадает с результатом индивидуального окрестностного осреднения каждой компоненты изображения. В этом случае обе процедуры осреднения дадут одинаковый результат. Однако такая эквивалентность реализуется не всегда, в частности нелинейные процедуры будут, как правило, давать различные результаты.

Псевдоцветные отображения. Когда монохромное изображение представлено в цветовом пространстве RGB и цветовые компоненты преобразуются независимо друг от друга, результат такого преобразования

называется псевдоцветным изображением, в котором входные полутоновые уровни заменены некоторыми цветами. Такие преобразования могут быть полезными, т. к. глаз человека способен различать миллионы цветов, но он отличает лишь относительно малое число оттенков серого тона. Поэтому псевдоцветные отображения часто используются при желании сделать малые колебания яркости монохромного изображения более заметными и различимыми для глаза и для выделения важных областей на полутоновых изображениях.

Следующий пример иллюстрирует сказанное. Преобразуем монохромное изображение сварного шва с дефектами в цветное, например в красное, следующими командами

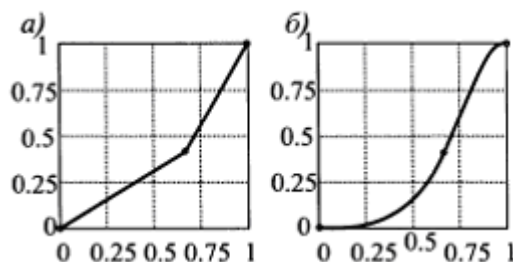
```
f=imread('Weld.tif'); imshow(f); % фото слева
fc=cat(3,double(f)/255,zeros(size(f)),zeros(size(f)));
imshow(fc); % фото в центре
z=double(f)/255;
zg=z.*(z>0.99); % создание зеленой компоненты
zb=0.5-0.25*z; % создание синей компоненты
g=cat(3,z,zg,zb);
imshow(g); % фото справа
```



Как видим, простая замена черно-белого монохромного изображения на монохромное красное мало что дает (рисунок в центре). Однако рисунок справа значительно качественнее представляет дефекты сварного шва. Для полноты впечатлений посмотрим, как выглядят созданные нами зеленая и синяя компоненты изображения сварного шва

```
imshow(zg); %зеленая компонента
'или в негативе imshow(imcomplement(zg));
imshow(zb); % синяя компонента
'или в негативе imshow(imcomplement(zb));
```

На следующем рисунке иллюстрируется простой, но весьма эффективный метод задания функций отображения графическим путем.



Задание функций отображения по контрольным точкам: а) – линейная интерполяция, б) – кубическая сплайновая интерполяция

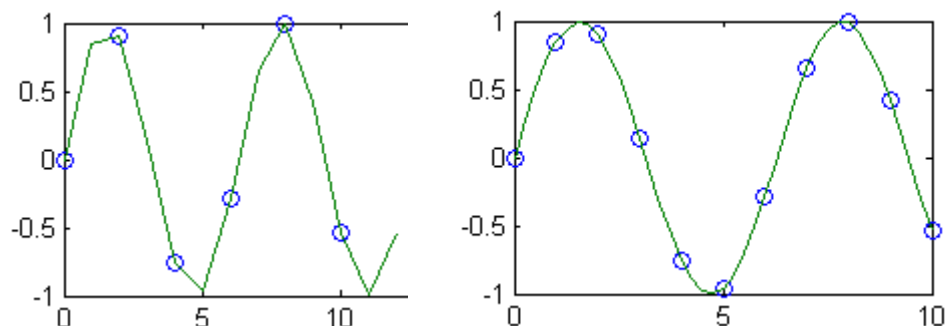


На рис. (а) показано преобразование, которое построено линейной интерполяцией по трем контрольным точкам (маленькие кружки на графике); рис.(б) показывает результат кубической сплайновой интерполяции по тем же трем точкам. Оба типа интерполяции реализованы в MATLAB. Линейная интерполяция совершается функцией

```
interp1q(x, y, xi);
```

которая возвращает значение ломаной, узловые точки которой задаются векторами (обычно вектор-столбцами)  $x$  и  $y$ , в точках вектора (вектор-столбца)  $xi$ . Элементы вектора  $x$  должны монотонно возрастать.

```
x=0:12; y=sin(x); % узлы ломаной
xi= [0 2 4 6 8 10];
yi=interp1(x,y,xi) % расчетные точки
yi = 0 0.9093 -0.7568 -0.2794 0.9894 -0.5440
plot(xi,yi,'o',x,y); % расчетные точки и ломаная (слева)
set(gcf,'Color',[1 1 1]);
```



Кубическая интерполяция выполняется функцией

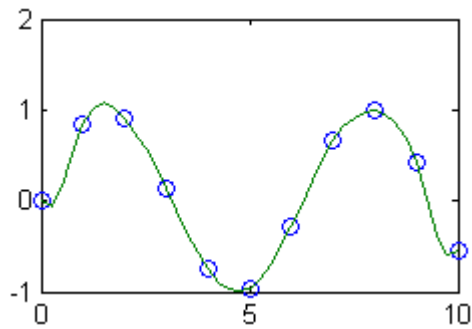
```
spline(x, y, xi),
```

аргументы которой имеют тот же смысл, что и выше.

```
x = 0:10; y = sin(x); % точки для интерполяции
xx = 0:.25:10;
yy = spline(x,y,xx); % точки на кубическом сплайне
% точки интерполяции и кубический сплайн (справа)
plot(x,y,'o',xx,yy); % рисунок выше справа
```

Точки  $xi$  должны быть различными. Кроме того, если в массиве  $y$  число элементов на 2 больше, чем в массиве  $x$ , то его первый и последний элементы должны быть коэффициентами наклонов касательных кубического сплайна в конечных точках. Например, добавление пары чисел к вектору  $y$  в его начале и конце изменит форму предыдущего кубического сплайна

```
x = 0:10; y = sin(x);
y2=[[-1],y,[1]]; % z=[z1,z2] % соединение вектор-строк z1 и z2
xx = 0:.25:10; yy = spline(x,y2,xx);
plot(x,y,'o',xx,yy);
```

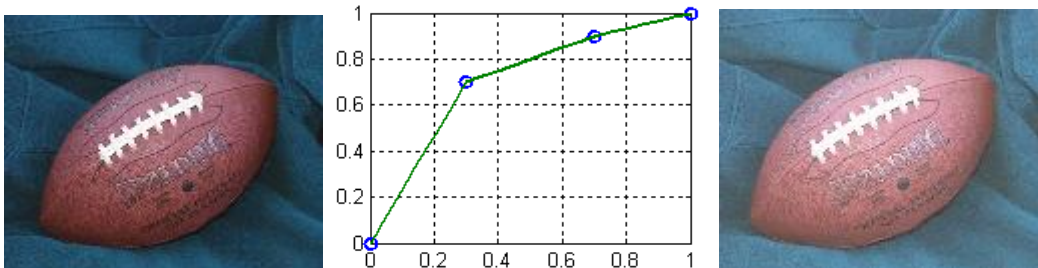


Использование этих функций может ускорить обработку изображений.

Создадим сценарий ex10.m

```
close all;
f=imread('football.jpg');
% f=imread('Chalk.tif');
% f=imread('peppers.png');
% f=imread('Mother.tif');
imshow(f); % фото слева
xp = [0 0.1 0.7 1];
yp = [0 0.3 0.9 1];
x=0:0.05:1; y=interp1(xp,yp,x);
figure,plot(xp,yp,'o',x,y,'LineWidth',2); % график в центре
grid on; set(gcf,'Color',[1 1 1]);
f1=double(f(:,:,1))/255;
f2=double(f(:,:,2))/255;
f3=double(f(:,:,3))/255;
z1=interp1(xp,yp,f1);
z2=interp1(xp,yp,f2);
z3=interp1(xp,yp,f3);
z=cat(3,z1,z2,z3);
figure,imshow(z); % изображение справа
```

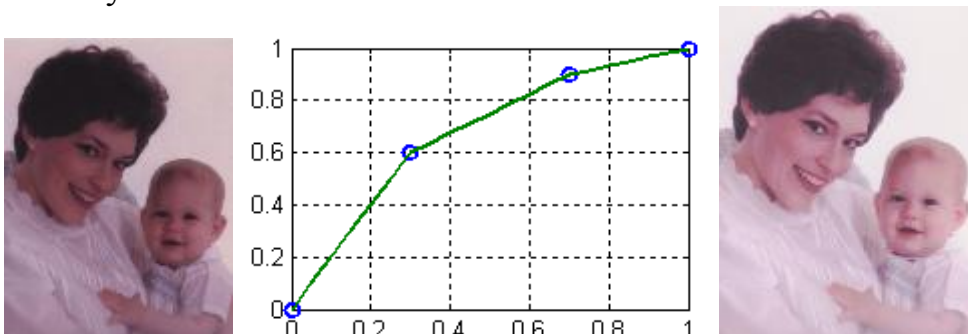
Он одинаково преобразует все три RGB плоскости изображения



Если в сценарии изменить три строки – имя файла исходного изображения и точки интерполяции, например, так

```
f=imread('Mother.tif');
xp = [0 0.3 0.7 1];
yp = [0 0.6 0.9 1];
```

то получим



Поскольку все три компонента обрабатывались одинаково, то преобладание розовой составляющей в изображении осталось, хотя контрастность стала значительно лучше.

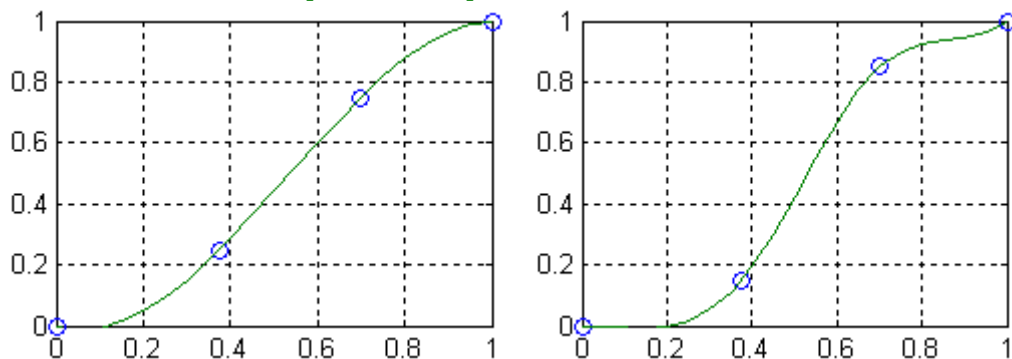
В следующем примере мы используем сплайновую интерполяцию для построения функций преобразования цветовой компонент изображения (сценарий ex12.m)

```
close all;
f=imread('Flowersbad.bmp'); imshow(f); % изображение слева
xp = [0 0.375 0.7 1];
yp=[0 0.25 0.75 1];
x=0:0.05:1; y=spline(xp,yp,x);
figure,plot(xp,yp,'o',x,y); % кривая слева
grid on; set(gcf,'Color',[1 1 1]);
axis([0 1 0 1]);

% Получаем цветовые double компоненты изображения
f1=double(f(:,:,1))/255;
f2=double(f(:,:,2))/255;
f3=double(f(:,:,3))/255;
% преобразуем компоненты в соответствии с кубическим сплайном
z1=spline(xp,yp,f1); z2=spline(xp,yp,f2); z3=spline(xp,yp,f3);
z=cat(3,z1,z2,z3); % собираем цветное изображение
figure,imshow(z); % изображение в центре

xp = [0 0.375 0.7 1];
yc=[0 0.15 0.85 1];
yp=[[0],yc,[1]]; % использование конечных касательных
x=0:0.05:1; y=spline(xp,yp,x);
figure,plot(xp,yc,'o',x,y); grid on; % кривая справа

z1=spline(xp,yp,f1); z2=spline(xp,yp,f2); z3=spline(xp,yp,f3);
z=cat(3,z1,z2,z3);
figure,imshow(z); % изображение справа
```



Графики функций преобразования



Исходное и преобразованные изображения

Заметим, что цветовые компоненты в этом примере преобразовывались одинаково.

Цветовые преобразования. Преобразования яркости – контрастности, такие как логарифмическое преобразование, полиномиальные, растяжения контрастности, кусочно-линейные, гистограммная эквализация, являются процессом преобразования полутоновых изображений. Поскольку цветные изображения состоят из нескольких компонент, монохромную процедуру необходимо приспособить к обработке более одной компоненты. Однако эти преобразования не следует делать независимо друг от друга, иначе можно получить ложные цвета. Правильный подход заключается в приближении к равномерному распределению для компоненты интенсивностей, оставляя сами цвета (компоненты цветового тона) неизменными. Поэтому весьма полезными могут быть процедуры преобразования изображения в форматы одна из составляющих которых является яркость (интенсивность) изображения. Тогда компоненту интенсивности можно преобразовать с помощью одной из рассмотренных процедур для монохромных изображений. Однако следует помнить, что MATLAB работает только с RGB изображениями. Поэтому потребуется обратное преобразование полученного изображения в формат RGB.

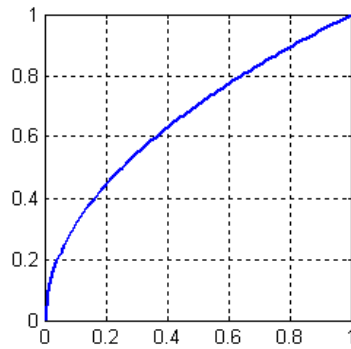
**Пример.** Преобразование в формате HIS.

```
f=imread('Caster.tif'); imshow(f); % фото слева
z=rgb2hsi(f); % преобразование в формат hsi
z1=z(:,:,1);z2=z(:,:,2);z3=z(:,:,3);
g3=imadjust(z3,[],[],0.5); % корректировка интенсивности I
g=cat(3,z1,z2,g3); % создание нового HSI изображения
e= hsi2rgb(g); % преобразование в формат RGB
figure,imshow(e); % изображение справа
```



Отметим, что функции преобразования из HSI в RGB формат `hsi2rgb(...)` и обратно `rgb2hsi(...)` не входят в стандартный пакет IPT, их код находится в нашем учебном каталоге.

График функции, с помощью которой мы корректировали интенсивность I, приведен на следующем рисунке.



Этот график построен с помощью следующего кода.

```
s=imadjust(0:0.01:1,[],[],0.5);
plot([0:0.01:1],s,'LineWidth',2); grid on;
set(gcf,'Color',[1 1 1]); axis([0 1 0 1]);
```

Следующий пример также демонстрирует полезность преобразования форматов изображения. На фото, показанном ниже слева, явно преобладает пурпурная компонента. Преобразуем его в формат СМУ и уменьшим пурпурную (М) компоненту.

```
f=imread('Mother.tif'); imshow(f); % фото слева
смy= imcomplement(f); % преобразуем в СМУ формат
z1=смy(:,:,1); z2=смy(:,:,2); z3=смy(:,:,3);
zz2=z2*0.85; % уменьшаем пурпурную компоненту
z=cat(3,z1,zz2,z3); % собираем СМУ изображение
g=imcomplement(z); % преобразуем в RGB формат
imshow(g); % изображение справа
```



**Пример. Балансировка цвета.** Вот пример различной обработки двух компонент СМУ изображения. Если последнее изображение преобразовать в СМУ формат и поработать с М (пурпурный) и Y (желтый) составляющими по – разному, то можно повысить контраст и снять преобладание розовой составляющей (сценарий ex11.m)

```
close all;
f=imread('Mother.tif');
imshow(f);
смy=imcomplement(f); % преобразуем в СМУ формат

xpm = [0 0.4 0.8 1];
yрm = [0 0.1 0.6 1];
x=0:0.05:1; y1=interp1(xpm,yрm,x);
figure,plot(xpm,yрm,'o',x,y1, 'LineWidth',2); % график слева
grid on; set(gcf,'Color',[1 1 1]);

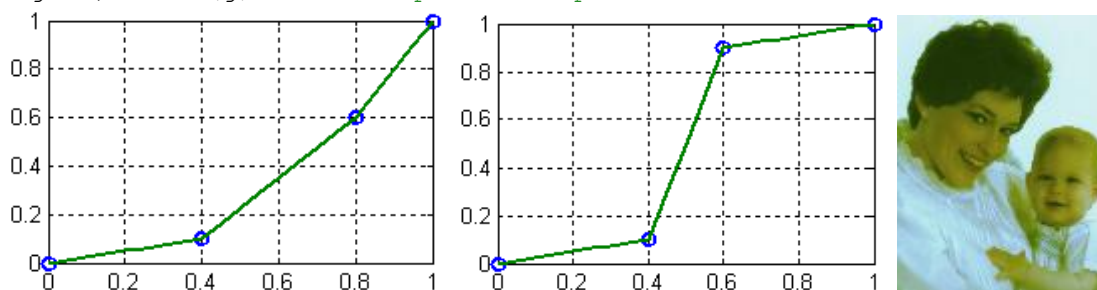
xру = [0 0.4 0.6 1];
yру = [0 0.1 0.9 1];
y2=interp1(xру,yру,x);
figure,plot(xру,yру,'o',x,y2, 'LineWidth',2); % график в центре
```

```

grid on; set(gcf,'Color',[1 1 1]);

f1=double(cmy(:,:,1))/255;
f2=double(cmy(:,:,2))/255;
f3=double(cmy(:,:,3))/255;
z2=interp1(xpm,ypm,f2);
z3=interp1(xpy,ypy,f3);
z=cat(3,f1,z2,z3);
g=imcomplement(z); % преобразуем в RGB формат
figure,imshow(g); % изображение справа

```



Преобразование M компоненты и Y компоненты

Отображения такого типа принято называть цветовой балансировкой или цветовой коррекцией. Раньше для выполнения таких действий требовались высококачественные цветопроизводящие системы, а теперь их можно совершить почти на любом персональном компьютере. Эта процедура используется при улучшении цветных фотографий. Хотя разбалансировку цвета можно установить объективно анализируя (с помощью цветного спектрометра) некоторый известный цвет на изображении, аккуратное визуальное оценивание также возможно, если на изображении имеются белые области, где RGB или CMY компоненты должны быть равны между собой. Оттенки человеческой кожи также могут служить отличным образцами для визуального оценивания. На компьютере мы можем выделить какую то часть изображения для которой цвет известен (например белый фон) проанализировать его и затем подобрать преобразование для всего изображения.

Пространственная фильтрация цветных изображений. Выше рассматривались преобразования цветных изображений, которые применяются к отдельным пикселям каждой цветовой плоскости (компоненты). Следующий уровень сложности соответствует обработке пространственных окрестностей, которая также совершается на каждой цветовой плоскости. Проиллюстрируем методы пространственной обработки цветных изображений на двух примерах процедур линейной фильтрации: сглаживание изображений и повышение резкости изображений.

Сглаживание (пространственная фильтрация) монохромных изображений осуществляется умножением значений всех пикселей в пространственной маске на соответствующие коэффициенты, суммированием и делением суммы на общее число элементов маски. Этот процесс (например, в пространстве RGB) определяется так же, как и при работе с монохромными изображениями, с той лишь разницей, что вместо

одиноким скалярным пикселем приходится работать с векторными величинами. Фактически одинаковая операция выполняется в каждой цветовой плоскости.

Линейные пространственные фильтры для сглаживания изображений строятся функцией `fspecial`, которая имеет три возможные опции: 'average', 'disk' и 'gaussian'. После того, как фильтр сгенерирован, процедура фильтрации совершается функцией `imfilter`.

В идейном плане сглаживание цветных RGB изображений линейным пространственным фильтром состоит из разделения изображения на цветовые компоненты, сглаживанием (фильтрацией) каждой или некоторых компонент, реконструкция отфильтрованного изображения. Однако можно совершать линейную фильтрацию прямо RGB изображений с помощью того же синтаксиса, что и при обработке монохромных изображений, что позволяет объединить описанные выше три шага в один.

```
fcfiltered = imfilter(fc,w);
```

Однако это делать не всегда целесообразно.

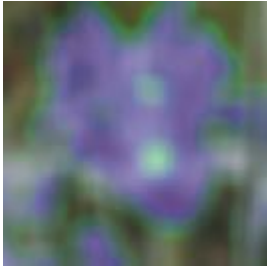
#### Пример.

```
f=imread('Flowersbad3.bmp'); imshow(f);  
h = rgb2hsi(f);  
H= h(:, :, 1); S = h(:, :, 2); I = h(:, :, 3);  
w = fspecial('average', 25);  
Ifiltered = imfilter(I, w, 'replicate');  
h = cat(3, H, S, Ifiltered);  
fc = hsi2rgb(h);  
imshow(fc);
```



Может показаться правильным сделать сглаживание по всем трем компонентам с помощью одного и того же фильтра. Однако в такой процедуре существенно изменятся соотношения между величинами цветового тона и насыщенности, что дает ложные цвета, как это видно на следующем рисунке

```
f=imread('Flowersbad3.bmp');  
h = rgb2hsi(f);  
H= h(:, :, 1); S = h(:, :, 2); I = h(:, :, 3);  
Hf = imfilter(H, w, 'replicate');  
Sf = imfilter(S, w, 'replicate');  
Ift = imfilter(I, w, 'replicate');  
h = cat(3, Hf, Sf, Ift);  
ff = hsi2rgb(h);  
imshow(ff);
```



Как видим, желтый цвет исчез.

Повышение резкости цветных изображений. Повышение резкости цветных RGB изображений с помощью линейного пространственного фильтра выполняется теми же процедурами, что и для монохромных изображений, но с использованием фильтра, повышающего резкость. Здесь этот метод демонстрируется на примере лапласиана. При этом лапласиан от цветного изображения можно находить с помощью вычисления лапласиана от каждой отдельной цветовой компоненты.

**Пример.** На следующем рисунке приведен слегка размытый вариант изображения цветка, полученного усредняющим фильтром размерами 5x5.

```
f=imread('Flowersbad3.bmp');
imshow(f);
```

Чтобы повысить резкость этого изображения, можно воспользоваться маской фильтра Лапласа. Затем улучшенное изображение (справа) вычисляется и отображается на дисплее командами

```
fen = imsubtract(f, imfilter(f, lapmask, 'replicate'));
imshow(fen); % изображение справа
```



Вот то же изображение, но еще более размытое и его восстановление с помощью фильтра Лапласа

```
close all;
f=imread('Flowersbad3.bmp');
h = rgb2hsi(f);
H= h(:,:,1); S = h(:,:,2); I = h(:,:,3);
w = fspecial('average', 5);
Ift = imfilter(I, w, 'replicate');
hh = cat(3, H, S, Ift);
ff = hsi2rgb(hh);
imshow(ff);
lapmask = [1 1 1; 1 -8 1; 1 1 1] ;
fen = imsubtract(ff, imfilter(ff, lapmask, 'replicate'));
figure, imshow(fen);
```



Здесь мы еще вычитаем одно изображение из другого. Само отфильтрованное изображение, получаемое командой `imfilter(ff, lapmask, 'replicate')` очень темное и неразлично.

Если усредняющий фильтр взять слишком большим, например, в строке 5 последнего сценария выполнить команду

```
w = fspecial('average', 25);
```

то фильтр Лапласа размером 3 x 3 конечно плохо справится со своей задачей

## **Литература.**

1. Р. Гонсалес, Р. Вудс Цифровая обработка изображений. М.: Техносфера, 2005.
2. Р. Гонсалес, Р. Вудс С. Эддинс Цифровая обработка изображений в среде Matlab. М.: Техносфера, 2006.
3. В.Т. Фисенко, Т.Ю. Фисенко, Компьютерная обработка и распознавание изображений: учеб. пособие. - СПб: СПбГУ ИТМО, 2008. –192 с.
4. Яне Б. Цифровая обработка изображений. Москва: Техносфера, 2007. - 584с.