Modeles_Couleurs

March 31, 2019

1 Modèles de couleurs

Marc Lorenzi
30 mars 2019

```
In [1]: import matplotlib.pyplot as plt
        import math, cmath
        import colorsys
        import random
```

```
In [2]: plt.rcParams['figure.figsize'] = (8, 8)
```

1.1 1. Les modèles de couleurs

1.2 2. Le modèle RGB

Le modèle RGB est le modèle utilisé pour décrire les couleurs affichées sur les écrans d'ordinateur. Il est utilisé par la plupart des fonctions de tracé de graphiques dans les langages de programmation "standard".

1.2.1 2.1 C'est quoi ?

Dans le modèle **RGB (Red, Green, Blue)** une couleur est codée par un triplet (r, g, b) de flottants entre 0 et 1, représentant les "quantités" de rouge, de vert et de bleu de la couleur considérée. Il s'agit d'un modèle *additif*, c'est à dire que le triplet $(0, 0, 0)$ représente la couleur noire et le triplet $(1, 1, 1)$ représente la couleur blanche. Le rouge vif est représenté par $(1, 0, 0)$, etc.

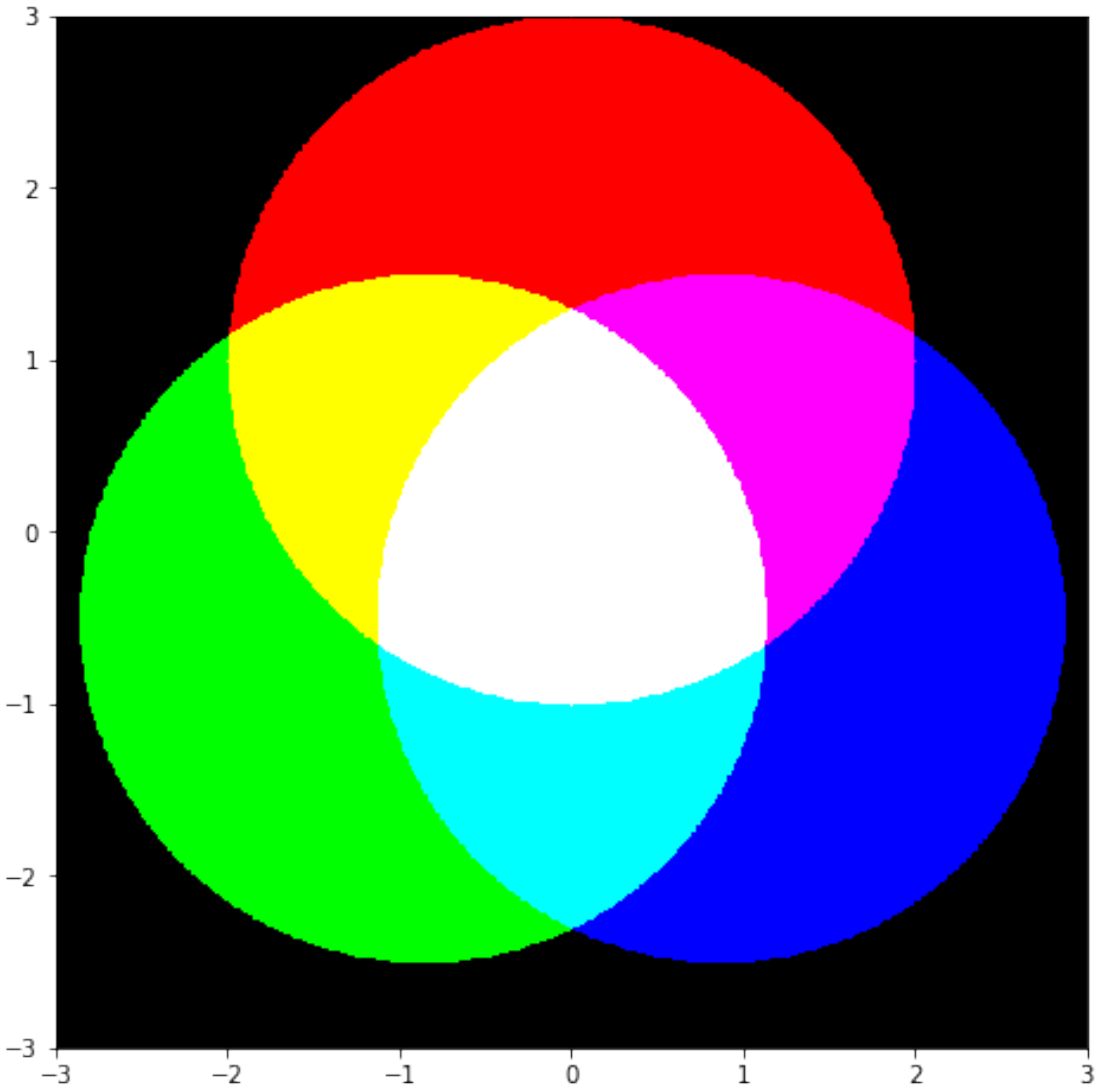
Dans le dessin ci-dessous, on représente trois disques colorés avec les couleurs primaires R, G, B. Les intersections de ces disques reçoivent la couleur obtenue en **additionnant** les couleurs des disques : le modèle RGB est un modèle **additif**, c'est ce que l'on obtient dans la réalité avec des objets qui émettent de la lumière, comme par exemple des spots lumineux.

```
In [3]: def distance(a, b, c, d):
            return math.sqrt((c - a) ** 2 + (d - b) ** 2)
```

```
In [4]: N = 600
        extent = [-3, 3, -3, 3]
        xmin, xmax, ymin, ymax = extent
        R = 2
        m = [[(0., 0., 0.) for i in range(N)] for j in range(N)]
```

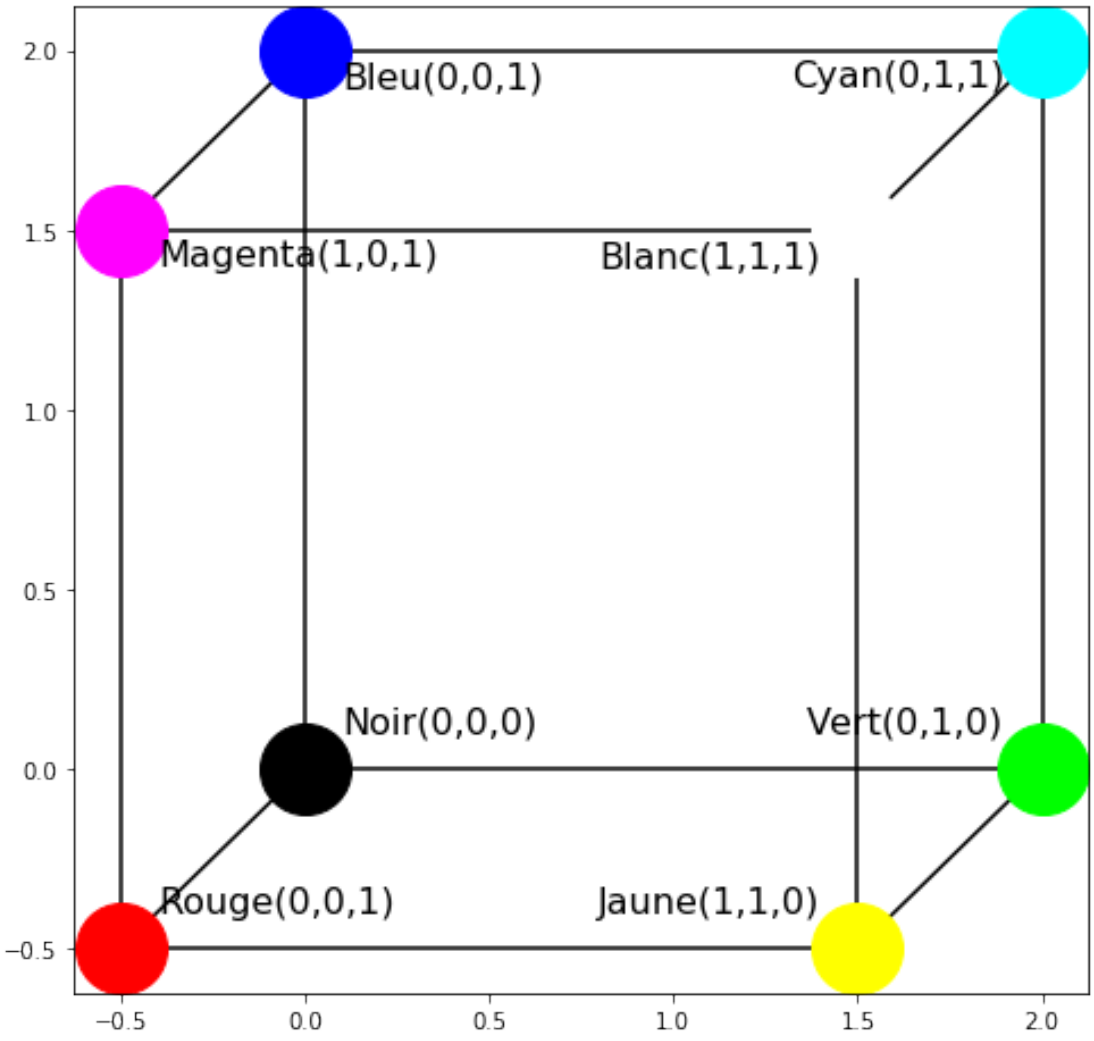
```
    for i in range(N):
        for j in range(N):
            r, g, b = 0., 0., 0.
            x = xmin + j * (xmax - xmin) / N
            y = ymax - i * (ymax - ymin) / N
            d = distance(x, y, math.sqrt(3)/2, -1/2)
            if d <= R: b = 1.
            d = distance(x, y, -math.sqrt(3)/2, -1/2)
            if d <= R: g = 1.
            d = distance(x, y, 0, 1)
            if d <= R: r = 1.
            m[i][j] = (r, g, b)
    plt.imshow(m, extent=extent)
```

Out[4]: <matplotlib.image.AxesImage at 0x1168780f0>



Autre représentation équivalente, le dessin ci-dessous représente le cube $[0,1] \times [0,1] \times [0,1]$. Les sommets du cube sont colorés en fonction de leurs coordonnées.

```
In [5]: plt.plot([0,-0.5],[0,-0.5],'k')
        plt.plot([-0.5,1.5],[-0.5,-0.5],'k')
        plt.plot([1.5,2],[-0.5,0],'k')
        plt.plot([2,0],[0,0],'k')

        plt.plot([0,-0.5],[2,1.5],'k')
        plt.plot([-0.5,1.5],[1.5,1.5],'k')
        plt.plot([1.5,2],[1.5,2],'k')
        plt.plot([2,0],[2,2],'k')

        plt.plot([-0.5, -0.5], [-0.5, 1.5], 'k')
        plt.plot([0, 0], [0, 2], 'k')
        plt.plot([1.5, 1.5], [-0.5, 1.5], 'k')
        plt.plot([2, 2], [0, 2], 'k')

        plt.plot([0],[0], 'o', color=(0,0,0), markersize=40)
        plt.plot([-0.5],[-0.5], 'o', color=(1,0,0), markersize=40)
        plt.plot([1.5],[-0.5], 'o', color=(1,1,0), markersize=40)
        plt.plot([2],[0], 'o', color=(0,1,0), markersize=40)

        plt.plot([0],[2], 'o', color=(0,0,1), markersize=40)
        plt.plot([-0.5],[1.5], 'o', color=(1,0,1), markersize=40)
        plt.plot([1.5],[1.5], 'o', color=(1,1,1), markersize=40)
        plt.plot([2],[2], 'o', color=(0,1,1), markersize=40)

        plt.text(-0.4, -0.4, "Rouge(0,0,1)", fontdict={'fontsize':16})
        plt.text(0.1, 0.1, "Noir(0,0,0)", fontdict={'fontsize':16})
        plt.text(-0.4, 1.4, "Magenta(1,0,1)", fontdict={'fontsize':16})
        plt.text(0.1, 1.9, "Bleu(0,0,1)", fontdict={'fontsize':16})
        plt.text(1.4, -0.4, "Jaune(1,1,0)", horizontalalignment='right', fontdict={'fontsize':16
        plt.text(1.9, 0.1, "Vert(0,1,0)", horizontalalignment='right', fontdict={'fontsize':16})
        plt.text(1.4, 1.4, "Blanc(1,1,1)", horizontalalignment='right', fontdict={'fontsize':16}
        plt.text(1.9, 1.9, "Cyan(0,1,1)", horizontalalignment='right', fontdict={'fontsize':16})

        plt.show()
```



Remarque : Imaginez que vous voyez le cube ci-dessus dans la direction de la diagonale Blanc → Noir. Les 6 sommets restants forment un hexagone. Dans le sens trigonométrique, vous voyez les couleurs Rouge, Jaune, Vert, Cyan, Bleu, Magenta. Bref, en gros les couleurs de l'arc en ciel. Nous y reviendrons lorsque nous parlerons du modèle HSV.

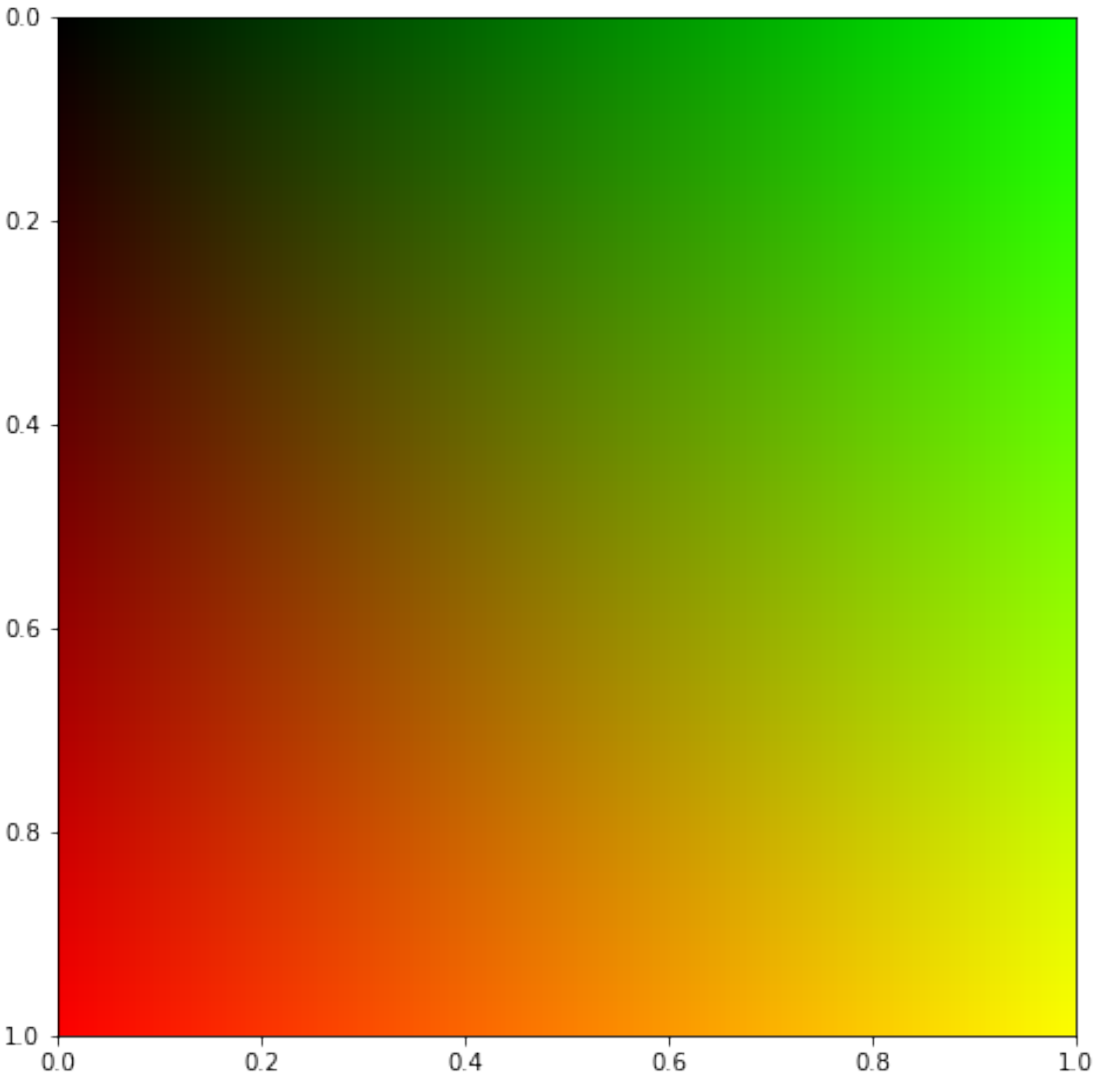
1.2.2 2.2 Affichons un carré coloré

Si le modèle RGB est très intuitif, il présente l'inconvénient d'être "tridimensionnel". Il n'est pas si évident que cela de faire un dessin dans lequels toutes les couleurs sont présentes. Le dessin ci-dessous, par exemple, affiche quelques couleurs dans un carré, mais pas toutes.

```
In [6]: N = 200
        m = [[0 for i in range(N)] for j in range(N)]
        for i in range(N):
            for j in range(N):
```

```
            m[i][j] = (i / N, j / N, 0)
    plt.imshow(m, extent=[0, 1, 1, 0])
```

```
Out[6]: <matplotlib.image.AxesImage at 0x113374f28>
```



En haut à gauche, nous avons $\frac{i}{N}$ et $\frac{j}{N}$ petits, et donc des couleurs proches de $(0,0,0)$, c'est à dire du noir. En bas à droite, nous avons i et j grands, et donc des couleurs proches de $(1,1,0)$, c'est à dire du jaune. Je vous laisse commenter les couleurs aux deux derniers coins de ce carré.

1.2.3 2.3 Barycentres !

Faisons une autre tentative. Soit (RGB) le triangle de sommets $R = (1,0)$, $B = (-1,0)$ et $G = (0,1)$. Tout point $P = (x,y)$ à l'intérieur de ce triangle est un barycentre de R, G et B à coefficients positifs :

$$P = \alpha R + \beta B + \gamma G$$

où $\alpha, \beta, \gamma \geq 0$ et pas tous nuls. Comme il s'agit d'un barycentre, ces coefficients ne sont pas déterminés de façon unique. Nous pouvons par exemple imposer $\alpha + \beta + \gamma = 1$. On a donc

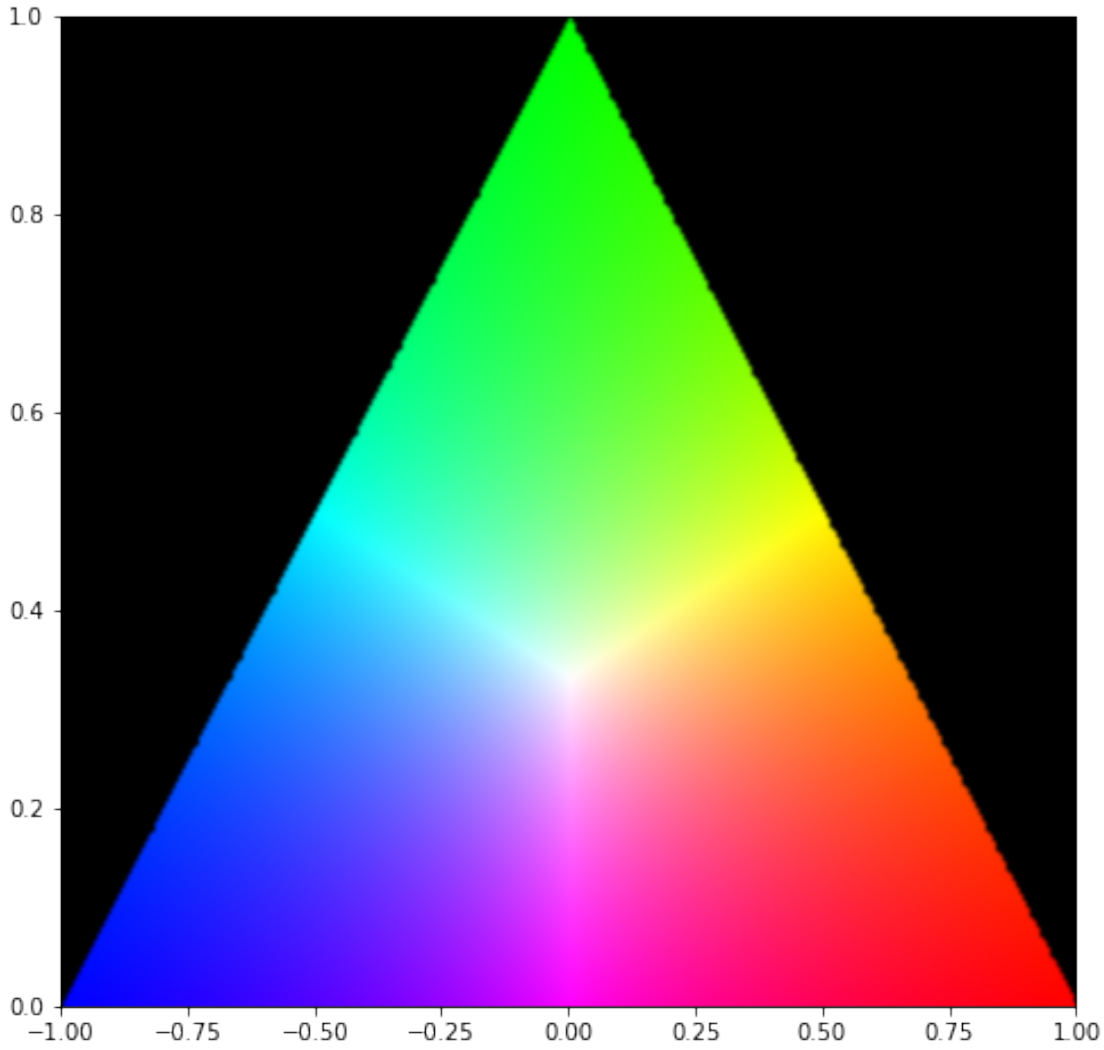
$$(x, y) = \alpha(1, 0) + \beta(-1, 0) + (1 - \alpha - \beta)(0, 1)$$

qui donne facilement

$$\alpha = \frac{x + 1 - y}{2} \quad \text{et} \quad \beta = \frac{-x + 1 - y}{2}$$

La cellule ci-dessous dessine le triangle coloré. Pour "chaque" point du rectangle $[-1, 1] \times [0, 1]$, on calcule les coefficients α, β, γ. S'ils sont tous les trois positifs, le point est dans le triangle et on le colorie en conséquence. Pour augmenter la luminosité du résultat, on divise α, β et γ par le plus grand de ces trois nombres. Cela amène l'une des trois valeurs au maximum, 1.

```
In [7]: N = 300
        m = [[(0, 0, 0) for i in range(N)] for j in range(N)]
        for i in range(N):
            for j in range(N):
                x = -1 + j * 2 / N
                y = 1 - i  / N
                a = (x + 1 - y) / 2
                b = (1 - y - x) / 2
                c = 1 - a - b
                d = max(a, b, c)
                if a >= 0 and b >= 0 and c >= 0:
                    m[i][j] = (a / d, c / d, b / d)
        plt.imshow(m, interpolation='bicubic', extent=[-1, 1, 0, 1], aspect='auto')

Out[7]: <matplotlib.image.AxesImage at 0x113d59cf8>
```



C'est un peu mieux que notre première tentative parce que nous arrivons à naviguer de façon continue entre le rouge, le vert et le bleu. Mais non, nous n'avons pas affiché toutes les couleurs possibles ! Passons à un autre modèle de couleurs, beaucoup plus intuitif à manipuler : le modèle HSV.

Remarque : Si vous tournez dans le sens trigonométrique autour du centre de gravité du triangle, vous parcourez les couleurs de l'arc en ciel.

1.3 3. Le modèle HSV

1.3.1 3.1 C'est quoi ?

Le modèle **HSV (Hue, Separation, Value)** est un modèle de couleurs plus intuitif à manipuler que le modèle RGB. Chaque couleur est représentée par un triplet (h, s, v) où

- h ("hue", **teinte** en français) est ce que les gens appellent communément la "couleur". $h = 0$ correspond au rouge. Lorsque h augmente, on passe par les couleurs de l'arc en ciel : rouge,

orange, jaune, vert, bleu, violet ... puis, lorsque h tend vers 1, on repasse du violet au rouge. Nous avons donc un cycle de couleurs. h est en gros le plus grand des trois paramètres r, g, b du modèle RGB, ou une combinaison des deux plus grands. Il nous faudra bien entendu être plus précis dans le suite.

- s est la **saturation**. plus s est petit, plus les couleurs sont "fades". Dans le système RGB, augmenter s consiste à augmenter le minimum des trois valeurs r, g, b.
- v ("value", "brightness", "luminance" ou "brillance" en français) est **l'intensité** de la couleur. Dans le système RGB, augmenter v consiste à augmenter simultanément r, g, b en gardant leurs proportions respectives constantes.

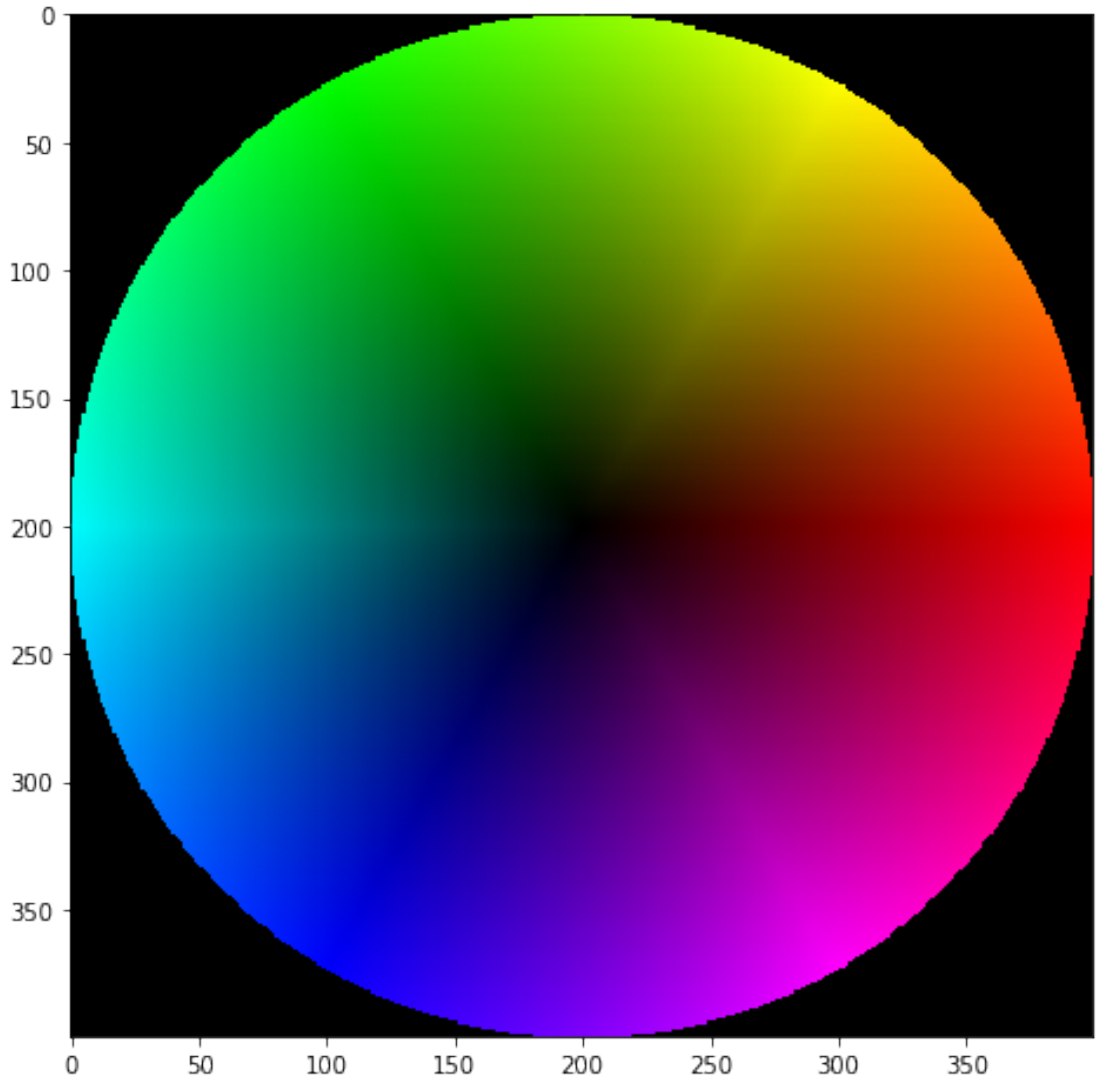
1.3.2 3.2 Un disque coloré

La cellule ci-dessous colorie les pixels d'un disque avec :

- Une intensité croissante du centre vers le bord.
- Une teinte vraiant comme l'angle polaire.

`matplotlib` atend des couleurs dans le modèle RGB. Utilisons pour l'instant le module `colorsys` pour faire des conversions.

```
In [8]: N = 400
        m = [[(0, 0, 0) for i in range(N)] for j in range(N)]
        for i in range(N):
            for j in range(N):
                x = -1 + j * 2 / N
                y = 1 - i * 2 / N
                z = x + 1j * y
                t = cmath.phase(z) / (2 * math.pi)
                if t < 0: t = t + 1
                r = abs(z)
                if r <= 1: m[i][j] = colorsys.hsv_to_rgb(t, 1, r)
        plt.imshow(m)

Out[8]: <matplotlib.image.AxesImage at 0x114576f28>
```



Et non, nous n'avons toujours pas affiché **toutes** les couleurs : comme nous avons pris $s = 1$ nous avons affiché toutes les couleurs de saturation maximale.

Exercice : Modifiez la cellule ci-dessus pour voir l'effet du paramètre s. Mettez-le à 0.5 au lieu de 1, par exemple. Qu'obtenez-vous pour $s = 0$? Eh oui, des couleurs vraiment très fades :-).

1.4 4. Conversions entre HSV et RGB

Soit (r, g, b) un triplet représentant une couleur dans le modèle RGB. Soit (h, s, v) sa représentation dans le modèle HSV. Comment passer d'une représentation à l'autre ?

1.4.1 4.1 RGB $\rightarrow$ HSV

Commençons par la conversion RGB $\rightarrow$ HSV. Les explications que j'ai données ci-dessus du modèle HSV étaient assez vagues. Ici, nous allons être plus précis. Notre but est d'écrire une fonction

`rgb_to_hsv` qui renvoie exactement les mêmes valeurs que la fonction correspondante du module `colorsys`. Pour cela, il faut évidemment se plonger dans la documentation, on ne peut pas **deviner** une convention ! Que nous dit-elle, cette doc ? On a tout d'abord

$$v = \max(r, g, b)$$

Posons ensuite

$$\Delta = \max(r, g, b) - \min(r, g, b)$$

Si $v = 0$, alors $r = g = b = 0$: nous avons affaire à du noir. On a donc $s = 0$. Sinon, on pose

$$s = \frac{\Delta}{v}$$

Dans la suite, on suppose $v > 0$. Passons au calcul délicat, celui de la teinte h. Si la saturation s est nulle, on a affaire à une teinte de gris. On prend $h = 0$. Sinon, $s > 0$ et donc, aussi, $D > 0$. La documentation du modèle HSV nous dit que trois cas se présentent :

- Cas 1, $r = v$. On pose $h_0 = \frac{g-b}{\Delta}$. Ici, on a $-1 \leq h_0 < 0$ si $g < b$ et $0 \leq h_0 < 1$ si $b \leq g$.
- Cas 2, $g = v$. On pose $h_0 = 2 + \frac{b-r}{\Delta}$. Dans ce cas $1 \leq h_0 < 3$.
- Cas 3, $b = v$. On pose $h_0 = 4 + \frac{r-g}{\Delta}$. On a dans ce cas $3 \leq h_0 < 5$.

Si, après calcul, $h_0 < 0$, on remplace h_0 par $h_0 + 6$. Enfin, on pose

$$h = \frac{h_0}{6}$$

Petit résumé utile pour la suite :

- Si $b \leq g \leq r$ alors $0 \leq h_0 < 1$.
- Si $b < r \leq g$ alors $1 \leq h_0 < 2$.
- Si $r \leq b \leq g$ alors $2 \leq h_0 < 3$.
- Si $r < g \leq b$ alors $3 \leq h_0 < 4$.
- Si $g \leq r \leq b$ alors $4 \leq h_0 < 5$.
- Si $g < b \leq r$ alors $5 \leq h_0 < 6$.

Remarque : Peut-on avoir $h = 1$? Apparemment non puisque $0 \leq h_0 < 6$, et donc $0 \leq h < 1$. Mais je préfère attendre la fin du prochain paragraphe pour développer un peu à ce sujet. Nous y verrons que $h = 1$ ne pose aucun problème.

```
In [9]: def rgb_to_hsv(r, g, b):
            v = max(r, g, b)
            D = v - min(r, g, b)
            if v == 0: s = 0
            else: s = D / v
            if s == 0: return (0, s, v)
            else:
                if r == v: h0 = (g - b) / D
                elif g == v: h0 = 2 + (b - r) / D
                else: h0 = 4 + (r - g) / D
                if h0 < 0: h0 = h0 + 6
                return (h0 / 6, s, v)
```

Testons avec des triplets (r, g, b) aléatoires. Comparons avec la fonction du module `colorsys`, nous devrions trouver des résultats identiques (aux erreurs d'arrondi près).

```
In [10]: r, g, b = (random.uniform(0, 1), random.uniform(0, 1), random.uniform(0, 1))
         print(rgb_to_hsv(r, g, b))
         print(colorsys.rgb_to_hsv(r, g, b))
```

```
(0.3891184695724062, 0.5935081875236976, 0.5345548785797065)
(0.3891184695724062, 0.5935081875236976, 0.5345548785797065)
```

Tout a l'air de fonctionner. Écrivons maintenant la fonction réciproque !

1.4.2 4.2 HSV $\to$ RGB

La conversion HSV $\to$ RGB est un peu plus délicate que la conversion inverse. Supposons dans ce qui suit h, s, v connus et cherchons r, g, b.

Si $s = 0$, c'est facile, on a affaire à une teinte de gris $r = g = b = v$. Supposons dorénavant $v > 0$. Reprenant les calculs du paragraphe précédent, on voit que

$$\Delta = vs$$

Rappelez-vous que $\Delta = \max(r, g, b) - \min(r, g, b)$.
Soit $h_0 = 6h$. Il y a 6 cas à considérer.

- Cas 0 : $0 \leq h_0 < 1$. On a donc $b \leq g \leq r = v$, $g - b = h_0\Delta$ et $\Delta = v - b$
- Cas 1 : $1 \leq h_0 < 2$. On a donc $b < r \leq g = v$, $b - r = (h_0 - 2)\Delta$ et $\Delta = v - b$.
- Cas 2 : $2 \leq h_0 < 3$. On a donc $r \leq b \leq g = v$, $b - r = (h_0 - 2)\Delta$ et $\Delta = v - r$.
- Cas 3 : $3 \leq h_0 < 4$. On a donc $r < g \leq b = v$, $r - g = (h_0 - 4)\Delta$ et $\Delta = v - r$.
- Cas 4 : $4 \leq h_0 < 5$. On a donc $g \leq r \leq b = v$, $r - g = (h_0 - 4)\Delta$ et $\Delta = v - g$.
- Cas 5 : $5 \leq h_0 < 6$. On a donc $g < b \leq r = v$, $g - b = (h_0 - 6)\Delta$ et $\Delta = v - g$.

On en déduit facilement les valeurs de r, g, b dans chacun des cas (exercice !)

- Cas 0 : $(r, g, b) = (v, v + (h_0 - 1)\Delta, v - \Delta)$.
- Cas 1 : $(r, g, b) = (v - (h_0 - 1)\Delta, v, v - \Delta)$.
- Cas 2 : $(r, g, b) = (v - \Delta, v, v + (h_0 - 3)\Delta)$.
- Cas 3 : $(r, g, b) = (v - \Delta, v - (h_0 - 3)\Delta, v)$.
- Cas 4 : $(r, g, b) = (v + (h_0 - 5)\Delta, v - \Delta, v)$.
- Cas 5 : $(r, g, b) = (v, v - \Delta, v - (h_0 - 5)\Delta)$.

D'où la fonction `hsv_to_rgb`.

```
In [11]: def hsv_to_rgb(h, s, v):
             if s == 0:
                 return (v, v, v)
             else:
                 D = v * s
                 h0 = 6 * h
                 if h0 < 1: return(v, v + (h0 - 1) * D, v - D)
```

```
        elif h0 < 2: return (v - (h0 - 1) * D, v, v - D)
        elif h0 < 3: return (v - D, v, v + (h0 - 3) * D)
        elif h0 < 4: return (v - D, v - (h0 - 3) * D, v)
        elif h0 < 5: return (v + (h0 - 5) * D, v - D, v)
        else: return (v, v - D, v - (h0 - 5) * D)
```

Testons tout cela.

```
In [12]: h, s, v = (random.uniform(0, 1), random.uniform(0, 1), random.uniform(0, 1))
         print(hsv_to_rgb(h, s, v))
         print(colorsys.hsv_to_rgb(h, s, v))
```

```
(0.16555734456982538, 0.3115688718416627, 0.005751668578433677)
(0.1655573445698254, 0.3115688718416627, 0.005751668578433694)
```

Autre test possible, composons `rgb_to_hsv` et `hsv_to_rgb`. On devrait trouver l'identité, aux erreurs d'arrondi près.

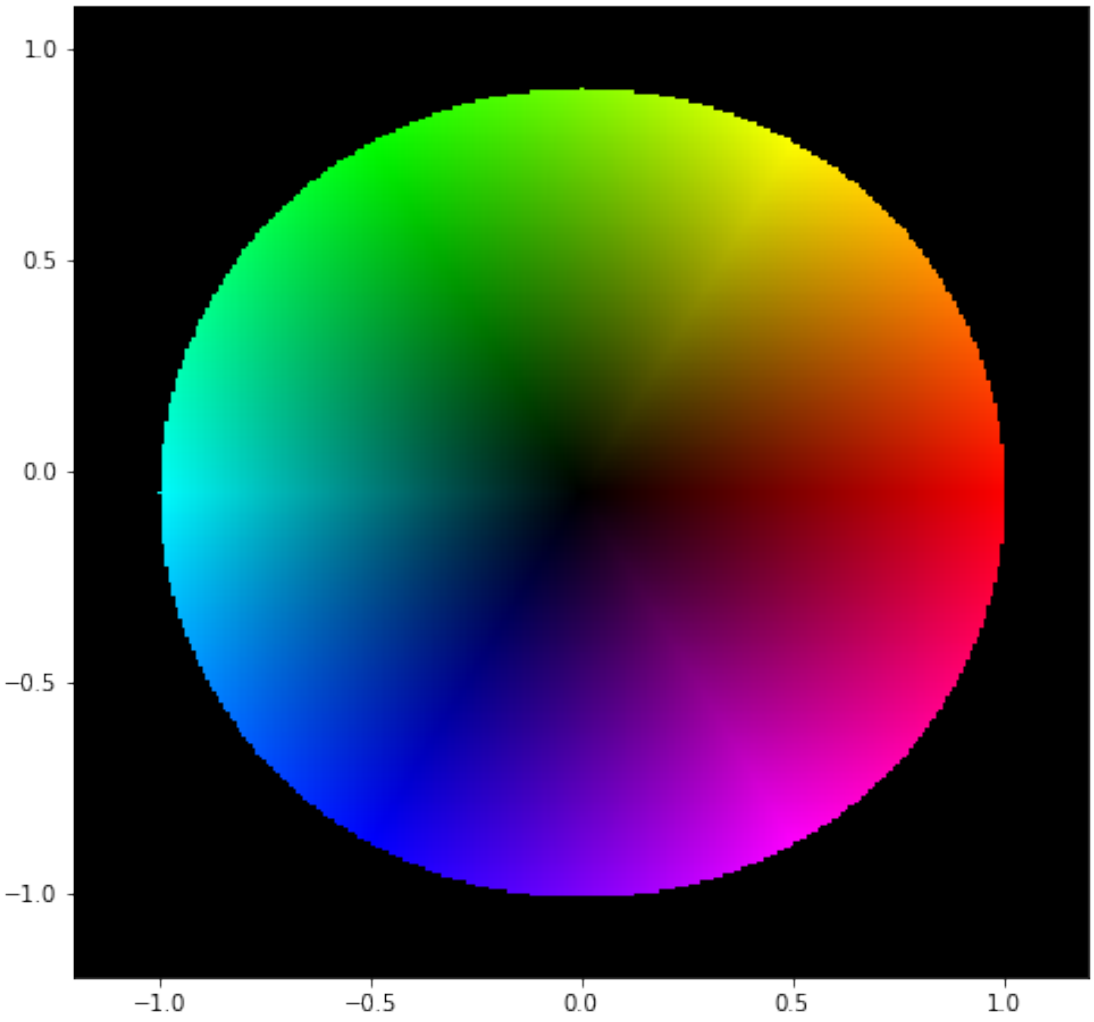
```
In [13]: h, s, v = (random.uniform(0, 1), random.uniform(0, 1), random.uniform(0, 1))
         r, g, b = hsv_to_rgb(h, s, v)
         h1, s1, v1 = rgb_to_hsv(r, g, b)
         print(h, s, v)
         print(h1, s1, v1)
```

```
0.3829182328472972 0.7055633994244755 0.5902587616822691
0.38291823284729726 0.7055633994244755 0.5902587616822691
```

Test ultime, redessinons le disque des couleurs que nous avions tracé un peu plus haut mais en utilisant notre propre fonction `hsv_to_rgb`.

```
In [14]: N = 300
         m = [[(0, 0, 0) for i in range(N)] for j in range(N)]
         for i in range(N):
             for j in range(N):
                 x = -1.2 + j * 2.4 / N
                 y = 1.2 - i * 2.4 / N
                 z = x + 1j * y
                 theta = cmath.phase(z)
                 if theta < 0: theta = theta + 2 * math.pi
                 r = abs(z)
                 if r <= 1: m[i][j] = hsv_to_rgb(theta / (2 * math.pi), 1, r)
         plt.imshow(m, extent=[-1.2, 1.2, -1.2, 1.1])
```

```
Out[14]: <matplotlib.image.AxesImage at 0x1117e2588>
```



1.4.3 4.3 Le cas $h = 1$

Nos formules de conversion RGB $\to$ HSV créent des triplets (h, s, v) tels que $0 \leq h < 1$. Qu'arrive t-il, juste pour voir, si on met le paramètre h à 1 dans la fonction réciproque `hsv_to_rgb` ?

Proposition : `hsv_to_rgb(0, s, v) = hsv_to_rgb(1, s, v)`.

Démonstration : Regardons le code de la fonction `hsv_to_rgb`.

- Si $h = 1$ alors $h_0 = 6$ et la fonction exécute la dernière branche `else` du test. Elle renvoie donc $(v, v - \Delta, v - (h_0 - 5)\Delta) = (v, v - \Delta, v - \Delta)$.
- Si $h = 0$, en revanche, on a $h_0 = 0$ et le premier test réussit. La fonction renvoie donc $(v, v + (h_0 - 1)\Delta, v - \Delta) = (v, v - \Delta, v - \Delta)$.

On obtient bien la même valeur dans les deux cas. En fait, plutôt que de voir h varier dans l'intervalle $[0, 1[$, il vaut mieux penser à h variant sur un cercle ...

Petite expérience avec le module `colorsys` ?

```
In [15]: r, g, b = colorsys.hsv_to_rgb(1, 0.3, 0.6)
         print(r, g, b)

0.6 0.42 0.42


In [16]: h, s, v = colorsys.rgb_to_hsv(r, g, b)
         print(h, s, v)

0.0 0.3 0.6
```

Eh oui, les fonctions `rgb_to_hsv` et `hsv_to_rgb` ne sont réciproques l'une de l'autre que si l'on suppose $0 = 1$:-).

Exercice : effacez `colorsys.` dans les deux cellules ci-dessus. Que font nos propres fonctions ?

1.5 5. Les modèles CMY et CMYK

1.5.1 5.1 Le modèle CMY

L'espace de couleurs CMY (Cyan, Magenta, Yellow) est l'ensemble des couleurs réalisables avec les trois couleurs Cyan, Magenta et Jaune. Contrairement à RGB, l'idée est que ces couleurs sont obtenues à l'aide de colorants (encre, peinture) et pas en "allumant" des pixels sur un écran. Qu'est-ce que cela change ? Eh bien la façon dont les couleurs s'additionnent.

Prenons un exemple.

- Une encre de couleur Cyan absorbe la lumière rouge et réfléchit la lumière bleue et la lumière verte.
- Une encre de couleur Jaune absorbe la lumière bleue et réfléchit la lumière rouge et la lumière verte.

Si on mélange à parts égales une encre Cyan et une encre Jaune, la lumière rouge et la lumière bleure seront absorbées, et seule la lumière verte sera réfléchie. Dans ce modèle on a donc Cyan + Jaune = Vert.

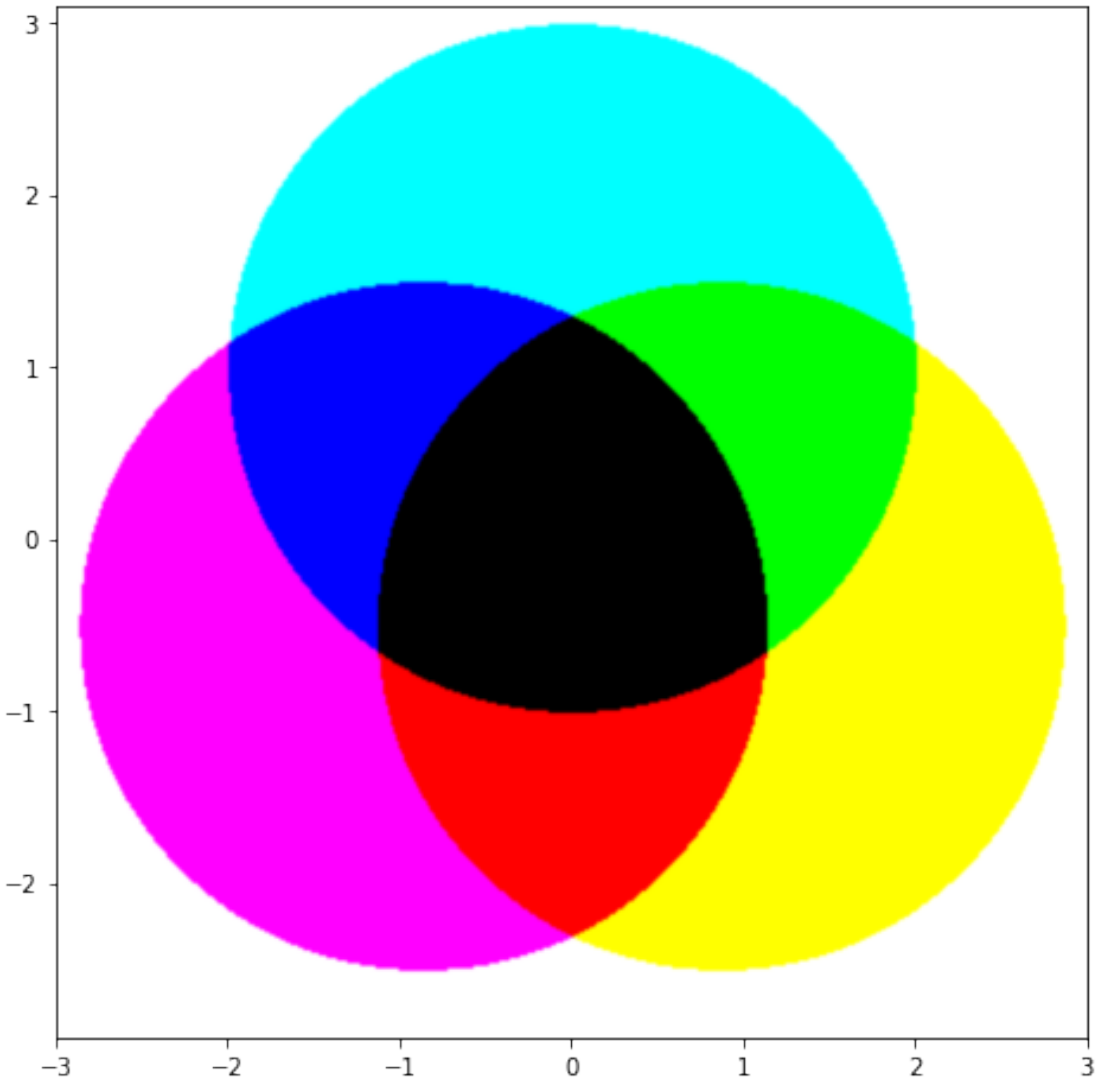
Dans le modèle RGB, en revanche, "allumer un pixel jaune" signifie en réalité allumer un pixel rouge et un pixel vert. Et "allumer un pixel Cyan" signifie allumer un pixel bleu et un pixel vert. Ainsi, "allumer" du jaune et du cyan donnerait Cyan + Jaune = Rouge + Vert + Vert + Bleu. Ce qui donnerait probablement du blanc ... enfin, là c'est un peu une discussion en l'air, parce qu'on ne peut pas additionner des triplets (r, v, b). Le cube $[0,1] \times [0,1] \times [0,1]$ n'est pas muni d'une loi de composition interne :-).

L'espace RGB est un espace où les couleurs **s'additionnent** (plus on allume de pixels plus la couleur est claire), alors que l'espace CMY est un espace où les couleurs **se soustrayent** (plus on mélange d'encres plus la couleur est sombre).

Voici un petit dessin qui est l'équivalent de ce que nous avons fait pour RGB. La différence est qu'au départ nous avons une feuille de papier blanche. En raisonnant dans le modèle RGB, tous les pixels de la feuille ont pour valeur $(r, g, b) = (1, 1, 1)$. Rajouter de l'encre sur la feuille a pour effet de diminuer les composantes des triplets. Ajouter de l'encre rouge, par exemple, transforme le pixel $(1, 1, 1)$ en le pixel $(0, 1, 1)$.

In [17]:
```
N = 400
extent = [-3, 3, -2.9, 3.1]
xmin, xmax, ymin, ymax = extent
R = 2
m = [[(1., 1., 1.) for i in range(N)] for j in range(N)]
for i in range(N):
    for j in range(N):
        r, g, b = 1., 1., 1.
        x = xmin + j * (xmax - xmin) / N
        y = ymax - i * (ymax - ymin) / N
        d = distance(x, y, math.sqrt(3)/2, -1/2)
        if d <= R: b = 0.
        d = distance(x, y, -math.sqrt(3)/2, -1/2)
        if d <= R: g = 0.
        d = distance(x, y, 0, 1)
        if d <= R: r = 0.
        m[i][j] = (r, g, b)
plt.imshow(m, extent=extent, interpolation='bicubic')
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x118c76588>
```



Remarquez que le noir est obtenu en mélangeant du Cyan, du Jaune et du Magenta à parts égales.

1.5.2 5.2 Le modèle CMYK

On peut théoriquement obtenir toutes les couleurs possibles, Noir inclus, à partir d'encres Cyan, Jaune et Magenta. Cela pose toutefois des problèmes pratiques. En effet, pour obtenir du noir il faut mélanger trois encres sur une zone du papier d'impression. Pour ne citer que quelques problèmes :

- Il est difficile d'obtenir de cette façon un noir vraiment noir. Il faudrait pour cela effectuer un mélange dans des proportions absolument parfaites.
- Le fait de déposer 3 encres sur la même zone rend le séchage plus long et fragilise le papier. Sur du papier de qualité médiocre cela peut même provoquer des déchirements.

- On frôle le non-sens lorsqu'on imprime des centaines de pages de texte en noir et blanc.
- Les encres de couleur coûtent plus cher que l'encre noire.

La solution retenue par la quasi-totalité des systèmes d'impression est de travailler avec 4 couleurs : Cyan, Magenta, Jaune, ET Noir. C'est le modèle CMYK. Pour créer une couleur donnée, on met autant de noir que possible, puis on ajuste les quantités de Cyan, Magenta et Jaune pour obtenir la couleur désirée.

Remarque : En langue française le modèle CMYK est appelé CMJN. Il est à la base du procédé d'impression par **quadrichromie**.

1.5.3 5.3 Conversions entre CMYK et CMY

Alors, combien met-on de noir ? Soit (c, m, y) le triplet représentant une couleur dans le système CMY. Soit (c', m', y', k) le quadruplet correspondant dans le système CMYK. Supposons par exemple $c \leq m \leq y$. L'idée est de prendre $k = c$, $c' = 0$, $m' = \lambda(m - k)$ et $y' = \lambda(y - k)$. On choisit λ pour que c', m' et y' soient entre 0 et 1. La valeur $\lambda = \frac{1}{1-k}$ fera l'affaire.

Notez le cas particulier $k = 1$, c'est à dire $c = m = y = 1$. Il correspond à la couleur noire.

Dans le cas général, on pose $k = \min(c, m, y)$. La fonction `cmy_to_cmyk` ci-dessous ne présente aucune difficulté.

```
In [18]: def cmy_to_cmyk(c, m, y):
             k = min(c, m, y)
             if k == 1: return (0, 0, 0, 1)
             else:
                 c1 = (c - k) / (1 - k)
                 m1 = (m - k) / (1 - k)
                 y1 = (y - k) / (1 - k)
                 return (c1, m1, y1, k)
```

La fonction réciproque est facile à obtenir. Je laisse mes vaillants lescteurs résoudrent des équations de degré 1 pour obtenir c, m, y en fonction de c', m', y', k.

```
In [19]: def cmyk_to_cmy(c, m, y, k):
             c1 = min(1, c * (1 - k) + k)
             m1 = min(1, m * (1 - k) + k)
             y1 = min(1, y * (1 - k) + k)
             return (c1, m1, y1)
```

Testons que nos fonctions sont bien réciproques l'une de l'autre ...

```
In [20]: c, m, y = random.uniform(0, 1), random.uniform(0, 1), random.uniform(0, 1)
         c1, m1, y1, k = cmy_to_cmyk(c, m, y)
         c2, m2, y2 = cmyk_to_cmy(c1, m1, y1, k)
         print(c, m, y)
         print(c2, m2, y2)
         print(c1, m1, y1, k)
```

```
0.009926636247547083 0.5476765622179441 0.8219360620342889
0.009926636247547083 0.5476765622179441 0.8219360620342889
0.0 0.5431414940124075 0.8201507640900112 0.009926636247547083
```

Reamrque : Remarquez que l'une des composantes c, m, y dans le modèle CMYK est nulle. Pour imprimer une couleur quelconque, on utilise du noir et DEUC autres couleurs. Dans la vraie vie la situation est un peu plus compliquée, mais je n'entrerai pas dans les détails.

1.5.4 5.4 Conversions entre CMY et RGB

Passer de CMY à RGB et vice-versa est immédiat.

```
In [21]: def rgb_to_cmy(r, g, b):
             c = 1 - r
             m = 1 - g
             y = 1 - b
             return (c, m, y)

In [22]: def cmy_to_rgb(c, m, y):
             r = 1 - c
             g = 1 - m
             b = 1 - y
             return (r, g, b)
```

Pour une fois je ne testerai pas mes fonctions :-).

1.5.5 5.5 Conversions entre CMYK et RGB

Il n'y a qu'à composer les fonctions déjà écrites :

- RGB $\rightarrow$ CMYK = RGB $\rightarrow$ CMY $\rightarrow$ CMYK
- CMYK $\rightarrow$ RGB = CMYK $\rightarrow$ CMY $\rightarrow$ RGB.

```
In [23]: def rgb_to_cmyk(r, g, b):
             c, m, y = rgb_to_cmy(r, g, b)
             return cmy_to_cmyk(c, m, y)

In [24]: def cmyk_to_rgb(c, m, y, k):
             c1, m1, y1 = cmyk_to_cmy(c, m, y, k)
             return cmy_to_rgb(c1, m1, y1)
```

Petit test ...

```
In [25]: r, g, b = random.uniform(0, 1), random.uniform(0, 1), random.uniform(0, 1)
         c, m, y, k = rgb_to_cmyk(r, g, b)
         r1, g1, b1 = cmyk_to_rgb(c, m, y, k)
         print(r, g, b)
         print(r1, g1, b1)
         print(c, m, y, k)
```

```
0.9107965694368867 0.6063880400305831 0.41878440034779696
0.9107965694368867 0.6063880400305831 0.41878440034779696
0.0 0.33422230564011524 0.5401998487909144 0.0892034305631133
```

1.5.6 5.6 Remarque finale : la réalité est plus compliquée

Dans la réalité les choses sont plus compliquées que cela. Chaque imprimante possède ses propres caractéristiques et ses propres encres. Il en résulte que la quantité d'encre de chaque couleur à choisir pour obtenir une couleur donnée n'est pas la même pour chaque imprimante. Certaines couleurs de l'espace total des couleurs sont même impossibles à obtenir par impression.

Aussi, il convient de calibrer l'imprimante. Très schématiquement, on imprime une feuille contenant des couleurs connues à l'avance puis on mesure à l'aide d'un photomètre quelles couleurs ont été réellement imprimées. À partir de ces mesures on crée un **profil** de l'imprimante.

Chaque imprimante est livrée avec un tel profil, un fichier dont l'extension est en général `.ICC` (International Color Consortium) ou `.ICN`. Le calcul réel du quadruplet (c, m, y, k) utilise les données contenues dans ce fichier afin que les couleurs imprimées soient aussi proches que possible des couleurs désirées.

Nos très jolies fonctions de conversion entre RGB et CMYK ne serviront donc à rien dans la pratique. Elles ne donnent que des estimations assez grossières.

Ces remarques sont également valables pour les moniteurs. Eh oui, même RGB est trompeur ! Chaque moniteur affiche les couleurs différemment. Et, donc, chaque moniteur est livré avec un profil `ICC` qui permet, étant donné un triplet (r, g, b) d'afficher sur l'écran la couleur qui approche le plus possible la "vraie" couleur (r, g, b).

```
In [ ]:
```