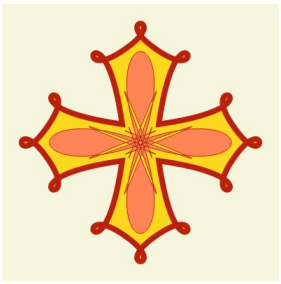
Vecteurs, translations et courbes de Bézier[1]

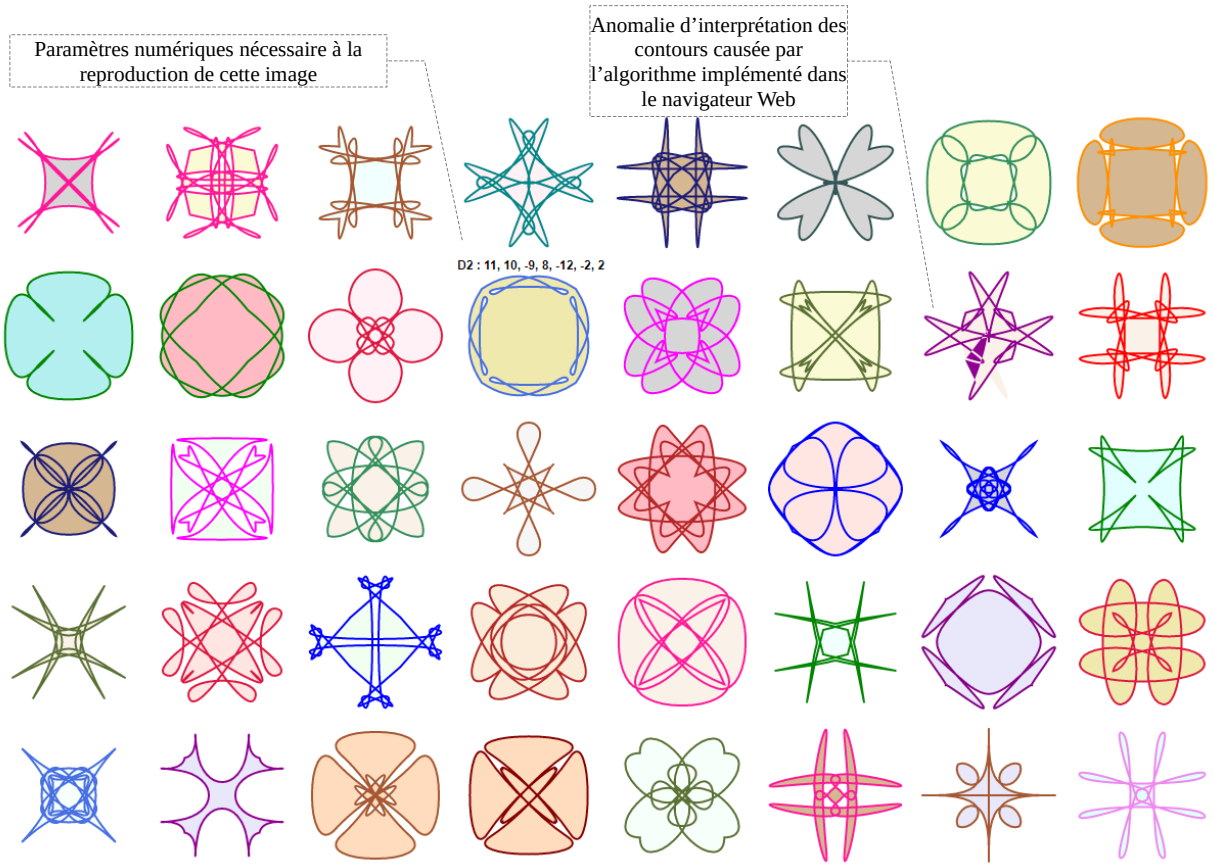
Activité de niveau seconde générale pour les Mathématiques et les Sciences Numériques et Technologie.



Les figures de cette page de présentation intègrent toutes exclusivement des courbes de Bézier : le contour de la croix occitane ci-contre et son motif interne ainsi que toutes les autres figures sont obtenues par un seul et même algorithme seuls les paramètres changent, il n'y en a que sept !

L'objectif de cette activité est de permettre à l'élève de comprendre la génération de telles figures et son lien avec les mathématiques et la technologie des images vectorielles du web.

Table des matières



Extrait d'une page web dont la partie svg du code a été produite à l'aide des algoritmes présentés dans cette activité

1 Courbes portant le nom de leur inventeur : Pierre Bézier (1910-1999) Ingénieur polyvalent français titulaire d'un doctorat, il a été formé dans les prestigieuses écoles d'Ingénieurs françaises des Arts et Métiers et Supélec.

Partie 1 : Expérimentations géométriques et découverte d'ensembles de points :

L'objectif de cette partie est d'aborder la notion d'ensembles de points obtenus à l'aide de translations de points mais c'est aussi l'occasion de se familiariser avec l'outil *geogebra (version classique 5)* de géométrie dynamique.

L'activité de recherche a proprement parler intervient à la fin de cette partie lorsque l'élève aura acquis les outils pour reproduire une figure familière (motif d'un jeu de carte : pique, trèfle, cœur ou carreau).

Du segment de droite à la courbe de Bézier d'ordre 2...

Soit A et B deux points distincts du plan. On note λ un réel.

À l'aide d'un logiciel de géométrie dynamique ou sous geogebra (version classique 5) :

- Placer deux points , les points A et B.
- Créer un curseur noté λ.
- Créer un point M image de A par translation de vecteur $\lambda\overrightarrow{AB}$ ainsi on a $M = t_{\lambda\overrightarrow{AB}}(A)$.

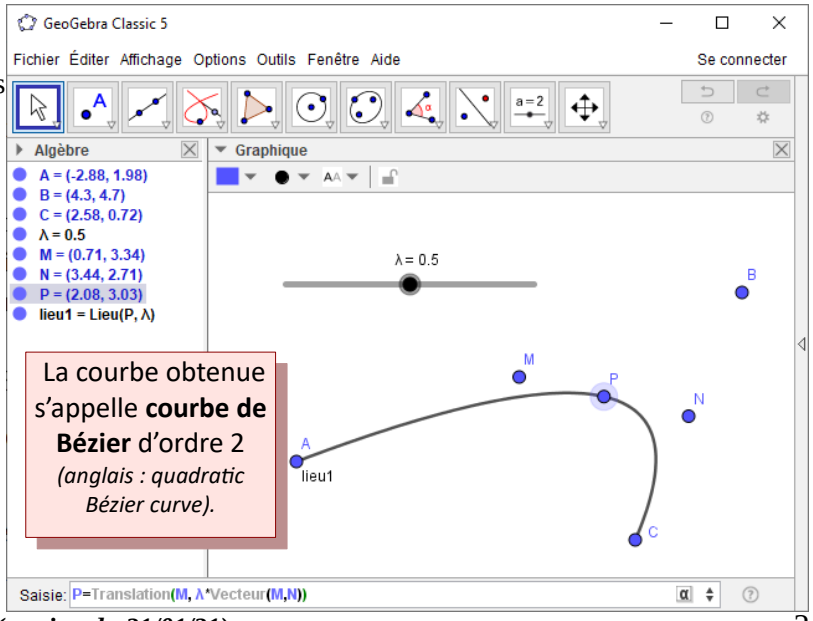
  Aide geogebra : entrer en ligne de saisie la formule : `M = Translation(A, λ*Vecteur(A, B))`
- Tracer le lieu de M lorsque λ varie, pour cela il faut utiliser l'outil lieu accessible via le menu lié au graphique.

1. À quel élément géométrique correspond l'ensemble des points M si λ décrit l'intervalle [0, 1] ?

2. Même question si λ décrit l'ensemble des réels ?

- Placer un point C tel que ABC soit un triangle.
- Créer un point N image de B par translation de vecteur $\lambda\overrightarrow{BC}$.
- Créer un point P image de M par translation de vecteur $\lambda\overrightarrow{MN}$.
- Tracer le lieu des points P lorsque λ décrit l'intervalle [0, 1].
- Tracer les vecteurs $\overrightarrow{AB}$ et $\overrightarrow{CB}$.

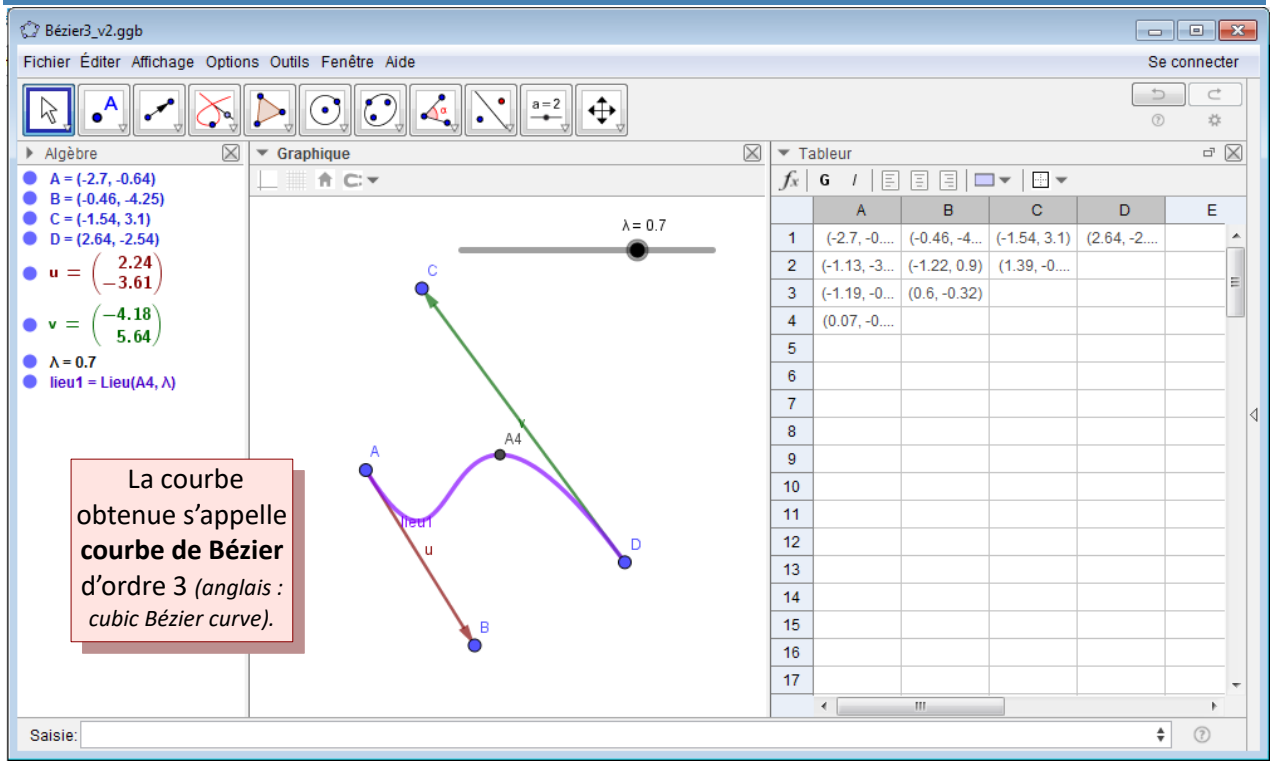
3. Faire varier la position du point B puis décrire « l'action » des vecteurs $\overrightarrow{AB}$ et $\overrightarrow{CB}$ sur la courbe « lieu des points P » (on pourra se référer à ce qui se passe au voisinage des points A et C).



de l'exploitation du tableur à la courbe de Bézier d'ordre 3…

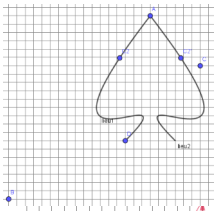
- Sauvegarder la dernière construction et dans une nouvelle construction placer 4 points deux à deux distincts notés A, B, C et D.
- Tracer les vecteurs $\overrightarrow{AB}$ et $\overrightarrow{DC}$.
- Créer un curseur noté λ pouvant varier entre 0 et 1
- Dans le tableur (sous Geogebra) saisir en cases A1, B1, C1 et D1 les points A, B, C et D. Cela se fait en saisissant en case A1 : `=A` etc. Sélectionner les quatre cases et décocher « afficher l'objet ».
- En ligne de saisie (sous Geogebra) saisir la formule :

  `A2=Translation(A1, λ*Vecteur(A1,B1)).`
- « Cliquer-décaler » la formule de la case A2 pour obtenir celle des cases B2, C2. Puis celle des cases A3 et B3 et de la même façon pour obtenir celle de la case A4.
- Tracer le lieu des points A4 lorsque λ décrit l'intervalle [0, 1].

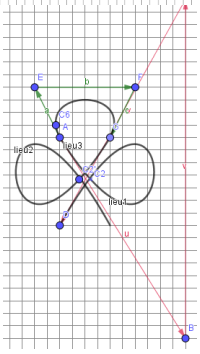


La courbe obtenue s'appelle **courbe de Bézier** d'ordre 3 *(anglais : cubic Bézier curve).*

4. Traduire en terme de translation comment est obtenu le point A4.



5. Déplacer les différents points initiaux, quel lien peut-on constater entre la courbe et les deux vecteurs $\overrightarrow{AB}$ et $\overrightarrow{DC}$? Que se passe-t-il sur la courbe lorsque ACBD est un parallélogramme ? Lorsque ACBD est un trapèze avec CB > AD ?



6. ***Proposer une construction de l'un des motifs d'un jeu de cartes (pique, trèfle, carreau ou cœur) à l'aide d'une ou deux courbes de Bézier initiales, d'un ou deux axes de symétrie et d'un quadrillage du plan pour en faciliter la reproduction (les courbes pique ou trèfle pourront rester ouvertes comme dans les exemples proposés pour en simplifier la conception) on résumera les paramètres utiles à la reproduction de ce motif.***

AÉR :

Partie 2 : Algorithme de translation .

L'objectif de cette partie est de mettre en place un algorithme de translation comme brique élémentaire pour l'élaboration de l'algorithme plus complexe donnant les coordonnées des points d'une courbe de Bézier.

L'étude effectuée dans la première partie à permis de mettre en exergue les paramètres utiles au placement d'un point sur une courbe de Bézier ainsi on a pu constater que l'ordre de la courbe de Bézier correspond au nombre de points de contrôle de cette courbe sans compter le premier point. On a également introduit un paramètre supplémentaire noté λ, réel décrivant l'intervalle [0, 1]. Ce réel permet de situer un point sur la courbe :

- λ = 0 correspond au point de « démarrage » de la courbe, c'est le premier point donné en paramètre.
- λ = 1 correspond au point terminal de la courbe, c'est le dernier point donné en paramètre.

On s'intéresse à la courbe de Bézier la plus répandue (notamment dans les images vectorielles du web), celle d'ordre 3.

1. Combien de nombres réels doit-on passer en argument à la fonction (ou algorithme) donnant les coordonnées d'un point de la courbe de Bézier d'ordre 3 ?

Soit $M_1(x_1, y_1)$ et $M_2(x_2, y_2)$ deux points du plan. On note $M(x_M, y_M)$ le point image de M_1 par translation de vecteur $\lambda \overrightarrow{M_1M_2}$ où λ est un réel de [0, 1].

2. Donner l'expression des coordonnées du vecteur $\overrightarrow{M_1M_2}$, en déduire que l'on peut écrire celles du point M par : $\begin{cases} x_M = x_1 + \lambda(x_2 - x_1) \\ y_M = y_1 + \lambda(y_2 - y_1) \end{cases}$.

3. Compléter le programme python suivant renvoyant les coordonnées du point M image de M_1 par translation de vecteur $\lambda \overrightarrow{M_1M_2}$ où λ est un réel de [0, 1] représenté par la variable `lbda`.

```
def translation(x1, y1, x2, y2, lbda):
    """Renvoie le point image de P1 par translation de vecteur lbda fois
    le vecteur M1M2 où M1(x1, y1) et M2(x2,y2)"""
    xM =
    yM =
    return xM, yM
```

Pour l'obtention d'un point de la courbe de Bézier on itère cette translation par couples de points successifs ce qui a été réalisé assez facilement avec un tableur (cf. partie 1) mais qui demande un peu d'astuce pour le traduire en programme.

Dans le tableur exploité lors de la première partie chaque ligne représente une génération de points.

4. Compléter le tableau ci-contre en précisant les translations effectuées pour chaque point proposé et son antécédent lorsque cela est nécessaire.

générations des points				
points initiaux	A	B	C	D
première génération	$A_1 = t_{\lambda \overrightarrow{AB}}(A)$	$B_1 = t_{\lambda \overrightarrow{BC}}(B)$	$C_1 = t_{\lambda}$ ( )	
deuxième génération	$A_2 = t_{\lambda \overrightarrow{A_1B_1}}(A_1)$	$B_2 = t_{\lambda}$ ( )		
point résultant	$A_3 = t_{\lambda}$ ( )			

Pour aller plus loin dans l'algorithme….*(les listes sont au programme de mathématiques de première mais tellement pratiques...)*

Dans le but d'automatiser le calcul des coordonnées de chaque point on commence par créer une liste de points. Sous python cela peut se faire en écrivant :
`L = [(xA, yA), (xB, yB), (xC, yC), (xD, yD)]`

En écrivant `L[0]` on accède au premier point[2] de la liste, c'est un couple[3] de deux nombres (l'abscisse et l'ordonnée du point A). Si l'on souhaite uniquement accéder à la valeur `xA` il faut utiliser un second niveau d'indexage car `xA` est dans la première case du premier couple de point ainsi on y accède en écrivant : `L[0][0]`.

5. Préciser comment accéder à la valeur `yC` sous python à partir de la liste L .

On propose le programme python suivant :

```
def BezierPt(xA, yA, xB, yB, xC, yC, xD, yD, lbda):
    """Renvoie les coordonnées du point de paramètre lambda (lbda)
    de la courbe de Bézier d'ordre 3 définie par les points A, B, C et D.
    A est le premier point (lbda=0) et D le dernier (lbda=1)"""
    L = [(xA, yA), (xB, yB), (xC, yC), (xD, yD)]
    for j in range(3):
        for k in range(3 - j):
            L[k] = translation(L[k][0], L[k][1], L[k+1][0], L[k+1][1], lbda)
        L.pop(-1) # retire le dernier élément de la liste L
    R = L[0] # R est le point résultant
    del(L) # supprime la liste L
    return R
```

6. Compléter le tableau suivant afin de vérifier que le programme proposé réponde bien à son commentaire :

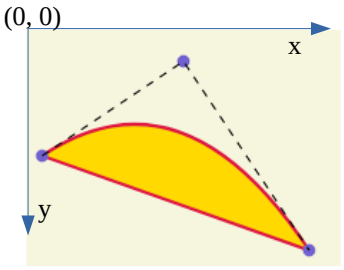
Nom de la variable :	Valeurs ou nom des variables et arguments obtenus à chaque appel à la fonction **`translation`** (pour la liste on gardera une notation symbolique par un nommage conforme à celui proposé à la question 4, les nouvelles valeurs renvoyées par l'exécution de la fonction **`translation`** sont surlignées en jaune)						
j	0	0	0	1	1		STOP !
k	0	1	2	0			
L	`[(xA,yA), (xB,yB), (xC,yC), (xD,yD)]`	`[(xA1,yA1) ,(xB,yB), (xC,yC), (xD,yD)]`	`[(xA1,yA1), (xB1,yB1), (xC,yC), (xD,yD)]`	`[(xA1,yA1), (xB1,yB1), (xC1,yC1)]`			
Arguments passés à la fonction **`translation`**	`xA, yA, xB, yB, lbda`	`xB, yB, xC, yC, lbda`	`xC, yC, xD, yD, lbda`				

7. Proposer une fonction Python renvoyant n points de la courbe de Bézier d'ordre 3 en exploitant les fonctions précédemment définies (on pourra choisir par défaut n=100).

2 Sous python les éléments d'une liste sont rangés par numéro de la case et la première case porte le numéro zéro.
3 Sous python ce couple est appelé un « tuple » c'est un objet qui ne peut être modifié mais qui peut être lu ou supprimé.

Partie 3 : Génération de codes de dessin vectoriel exploitant les courbes de Bézier (SNT).

Le web permet l'accès à des images de différents formats de codage il en existe principalement deux familles : les **images matricielles** qui peuvent être immédiatement affichées (après une éventuelle décompression) et les **images vectorielles** qui ne sont que des codes contenant les paramètres indispensables aux algorithmes de restitution de ces images de façon à adapter la résolution de ces images au système de restitution (écran, imprimante...). Ainsi les images vectorielles ont l'avantage de paraître toujours « au top » pour le système de restitution et il n'y a pas d'effet de pixellisation par zoom sur l'image puisqu'à chaque fois cette image vectorielle est recalculée.



Pour tracer une courbe de Bézier d'ordre 2 dans un dessin vectoriel SVG[4] (Scalable Vector Graphics) on a besoin de 6 nombres (correspondant aux coordonnées des 3 points de contrôle de cette courbe cf. partie1) et d'un codage indiquant qu'on est en présence d'une telle courbe, cela donne l'exemple de code : `d="M 10,80 Q 100,20 180,140 z"` *(z pour fermer la courbe)*

Q (Quadratic Bézier) : on réalise un chemin en suivant une courbe de Bézier d'ordre 2 depuis le point courant exploitant les deux points suivants donnés ici en coordonnées absolues

M (Moveto) : on se positionne au point de coordonnées (10, 80)

Définition du chemin

<u>Attention :</u> L'axe des ordonnées du repère utilisé en imagerie est orienté vers le bas !

Ce code peut être intégré directement dans une page html sous une balise svg (on y a ajouté d'autres éléments non nécessaires au tracé mais permettant de faire apparaître les points de contrôle et les segments) :

```
<!DOCTYPE html>
<html>
	<body>
		<!-- balise du SVG indiquant les dimensions en pixels du dessin vectoriel -->
		<svg height="150" width="200" style="background-color:beige">
			<!-- définition d'un chemin avec ses paramètres -->
			<path stroke-width="2" stroke="Crimson" fill="gold"
			d="M 10,80 Q 100,20 180,140 z" />
			<!-- Tracé des points -->
			<g fill="SlateBlue">
				<circle cx="10" cy="80" r="4"  />
				<circle cx="100" cy="20" r="4" />
				<circle cx="180" cy="140" r="4" />
			</g>
			<!-- Tracé des segments -->
			<path stroke="black" stroke-width="1" fill="none" stroke-dasharray="5,5"
			d="M10,80 L 100,20 180,140" />
		</svg>
	</body>
</html>
```

On donne le code suivant : `<path stroke="none" fill="green"`
`d="M 60,140 c 80,-220 140,20 20,-60 c -80,-100 80,-100 0,0 c -120,80 -60,-160 20,60 Z" />`

c (cubic Bézier) : on réalise un chemin en suivant une courbe de Bézier d'ordre 3 depuis le point courant exploitant les trois points suivants qui sont ici donnés en coordonnées relatives (au premier point)

1. Copier ce code dans une balise svg aux dimensions 160 pixels de large par 150 pixels de haut et l'intégrer dans un fichier html, quelle figure cela produit-il ? Tous les points de contrôle apparaissant dans le code de cette figure peuvent-ils être positionnés dans le cadre prévu?

4 Format de dessin vectoriel défini par l'organisme international de normalisation du Web le W3C et compatible avec la norme HTML5 ainsi restituable par les derniers navigateurs Web.

On souhaite faciliter la génération de courbes de Bézier en code SVG à l'aide d'un programme python. Dans la partie mathématique on est parvenu à calculer les coordonnées des points d'une courbe Bézier d'ordre 3, une version améliorée de ce programme permet le calcul de points d'une courbe de Bézier d'ordre *od*[5] (les fonctions python exploitées sont données en annexe). Pour intégrer au mieux une courbe de Bézier dans une image vectorielle il faut être capable d'en déterminer la zone occupée, cela est réalisé par la fonction `CadreBezier`. On notera que les coordonnées des points de contrôles ne suffisent pas pour établir ce cadrage, en effet, ils peuvent sortir de la zone de dessin sans que la figure n'en soit déformée.

On donne la liste des points de contrôle d'une courbe de Bézier dans un repère orthonormé standard :

```
L = [ (0, 5), (2, -2), (7, -2), (1, -3) ]
```

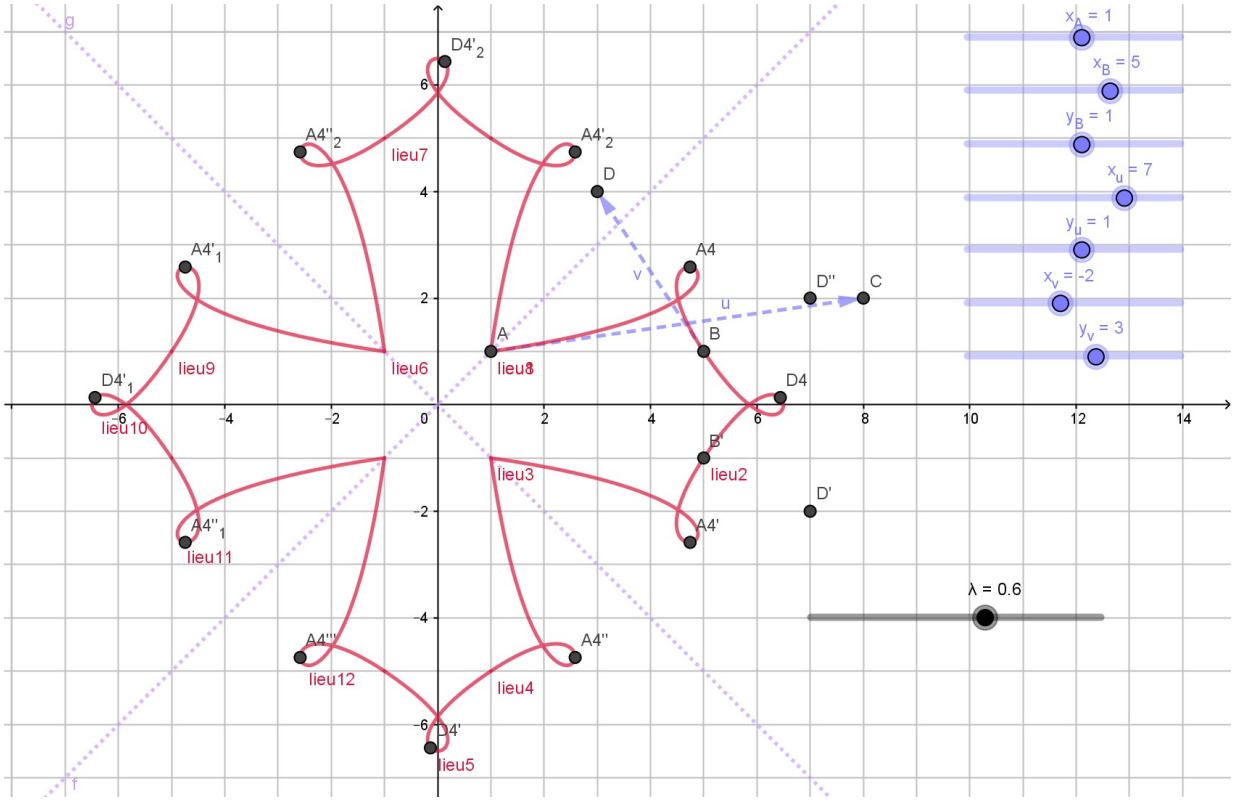
2. Indiquer de quel est l'ordre de la courbe de Bézier associée à L, puis en exploitant la fonction `CadreBezier`, déterminer la zone du plan occupée par celle-ci. On comparera les résultats obtenus suivant que `entier=True` ou `entier=False`, on indiquera quel point sort du cadre et on pourra contrôler ces résultats à l'aide de geogebra avec le fichier **`Bezier_ordre3_quadrillage.ggb`**.

On souhaite que la courbe associée à L s'affiche dans un cadre carré de coté 400 pixels d'une zone de dessin vectoriel (SVG) de largeur 800 pixels et de hauteur 400 pixels avec une marge de 30 pixels (marge interne de type : « padding »).

3. Exploiter le programme python `BezierSvgN` fourni pour produire le code SVG de cette image et l'afficher dans une page web.

On a voulu créer un logo présentant une croix occitane représentée ci-dessus, pour ce faire il n'a fallut que 7 paramètres (7 nombres réels) :

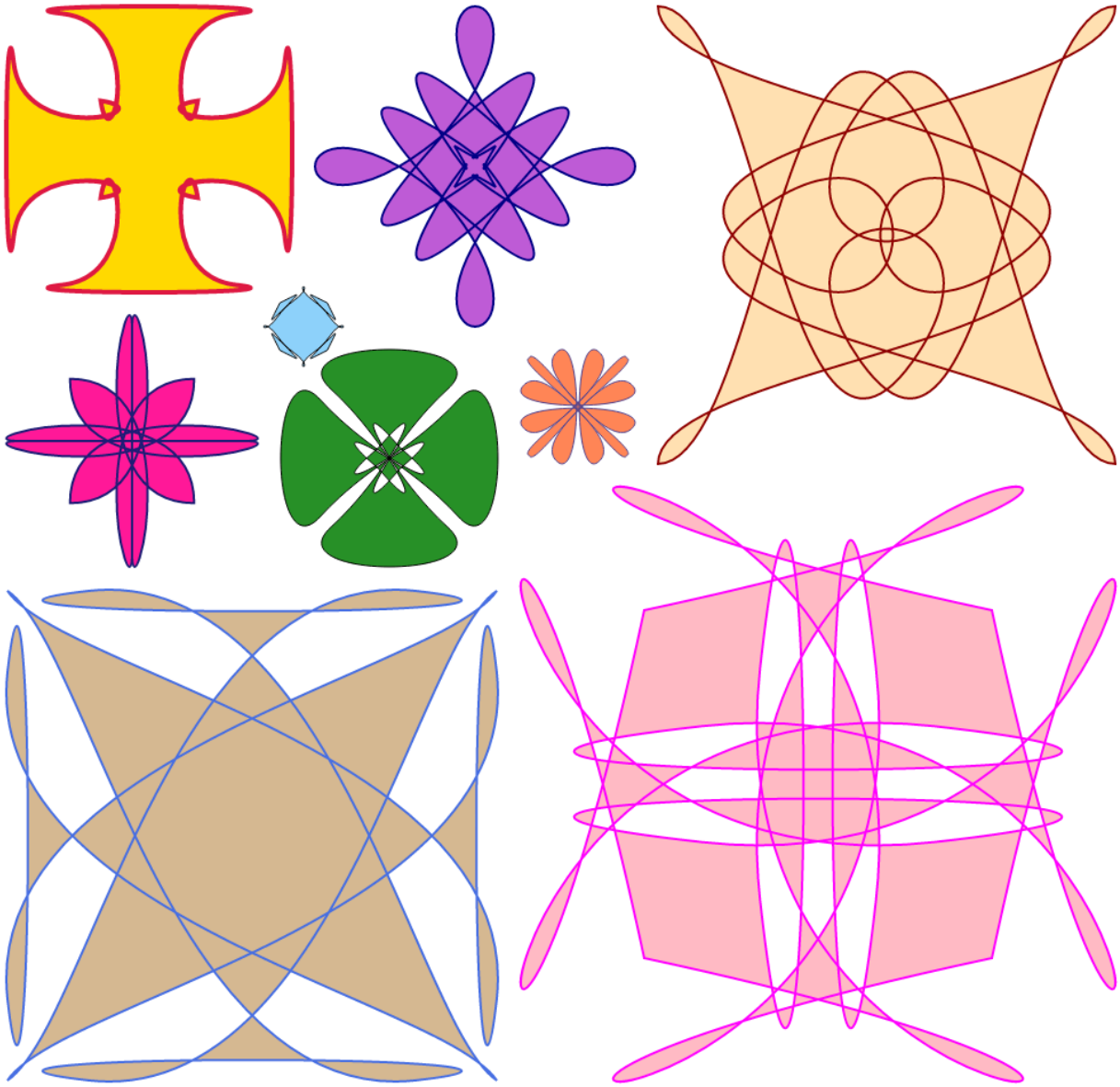
- ✗ L'abscisse du point A
- ✗ Les coordonnées du point B
- ✗ Les coordonnées des vecteurs $\vec{u}$ et $\vec{v}$ (ou les coordonnées des points C et D).



5 En pratique on se limitera aux deux courbes de Bézier acceptés par le SVG : od=2 (Bézier d'ordre 2) ou od=3 mais le programme permet le calcul de courbes de Bézier d'ordre supérieur à 3.

4. En observant la figure proposée décrire les symétries à envisager et le nombre de courbes de Bézier nécessaires pour pouvoir la reproduire.

L'algorithme permettant la production de la croix occitane précédente permet également l'obtention d'un nombre infini d'autres figures bâties sur la base de deux courbes[6] de Bézier d'ordre 3 et présentant les 4 axes de symétrie. Quelques échantillons de ces figures sont fournis sur l'image suivante et dans la page d'introduction.



5. Exploiter les programmes python proposés pour produire une image SVG d'une figure 2B3_S4[7] de votre choix parmi celles proposées dans l'une des planches données sur l'espace pédagogique SNT/Thème image/SVG. Cette figure devra pouvoir s'insérer dans un cadre carré de 400 pixels de coté en préservant une marge de 30 pixels.

6 Les paramètres de la seconde courbe étant totalement contraints par la présence d'axes de symétrie et par le fait qu'elle démarre avec le vecteur directeur opposé à celui terminant la première courbe : $-\vec{v}$.

7 Cette notation se lit : figure élaborée par 2 courbes génératrices de Bézier d'ordre 3 (2B3) et 4 axes de symétrie (S4)

Annexe : codes python des fonctions de base

```
def translat(P1, P2, lbda):
    """Renvoie le point image de P1 par translation de vecteur lbda fois
    le vecteur P1P2 où P1 et P2 sont des tuples : (x1,y1) et (x2,y2)."""
    L=[u + lbda*(v-u) for (u,v) in zip(P1,P2)]
    return L[0], L[1]


def bezierNPt(L, lbda):
    """Renvoie les coordonnées du point de la courbe de Bézier d'ordre n
    définie par la liste L des points au paramètre lambda (lbda)
    L est une liste de couples de coordonnées"""
    Ls = [k for k in L] # Reproduit la liste pour ne pas écraser l'originale
    assert len(Ls)>1 and len(Ls[0])==2 , "erreur dans la liste L"
    od = len(Ls) - 1 # L'ordre est donné par le nb de pts de contrôle - 1
    for j in range(od):
        for k in range(od-j):
            Ls[k] = translat(Ls[k], Ls[k+1], lbda)
        Ls.pop(-1) # retire le dernier élément de la liste Ls
    R = Ls[0] # R est le point résultant
    del(Ls) # Supprime la liste temporaire
    return R


def CadreBezier(L, n=200):
    """Renvoie les dimensions du cadre dans lequel se situe la courbe
    de Bézier définie par les points de contrôle de la liste L.
    Pour cela on calcule n points de la courbe"""
    xMin, yMin = L[0]
    xMax, yMax = L[0]
    for k in range(1, n+1):
        t = k/n
        xB, yB = bezierNPt(L, t)
        if xB < xMin :
            xMin = xB
        elif xB > xMax:
            xMax = xB
        if yB < yMin:
            yMin = yB
        elif yB > yMax:
            yMax = yB
    return xMin, yMin, xMax, yMax


def CadreBS4(L,n=200):
    """Renvoie les dimensions du cadre dans lequel se situe une figure
    2B3S4 définie par les paramètres de la liste L.
    Pour cela on calcule n points de la courbe. Si entier=False les
    dimensions du cadre ne sont pas arrondies"""
    Ls = [(L[0],L[0]),(L[0]+L[3],L[0]+L[4]),(L[1]+L[5],L[2]+L[6]),(L[1],L[2])]
    Lcmax = CadreBezier(Ls,n)
    cmax1 = max(abs(k) for k in Lcmax)
    Ls = [(L[1],L[2]),(L[1]-L[5],L[2]-L[6]),(L[1]-L[5],L[6]-L[2]),(L[1],-L[2])]
    Lcmax = CadreBezier(Ls,n)
    cmax2 = max(abs(k) for k in Lcmax)
    cmax = max(cmax1,cmax2)
    return cmax


def PFigS4ToListePts(L):
    """Renvoie la listes des points primitifs d'une figure S4 de paramètres
    situés dans la liste L de type :[xA, xB, yB, xu, yu, xv, yv]"""
    Dep = [(L[0],L[0])]
    Ls = [(L[0]+L[3],L[0]+L[4]),(L[1]+L[5],L[2]+L[6]),(L[1],L[2]),
          (L[1]-L[5],L[6]-L[2]),(L[1],-L[2]),
          (L[0]+L[3],-L[0]-L[4]),(L[0],-L[0])]
    Lc = [k for k in Ls]
    for k in range(3):
        Lc = [(v,-u) for (u,v) in Lc]
        Ls += Lc
    del(Lc)
    return Dep+Ls
```

```python
def BezierSvgN(L, t, x0=0, y0=0, mg=15):
    """Renvoie le code SVG pour l'affichage d'une courbe de Bézier
    de points définis dans la liste L de coordonnées (L est une
    liste de tuples des coordonnées des points).
    Met la courbe à l'échelle pour intégrer un carré de côte t en
    préservant une marge mg avec l'un des bords et la positionne
    aux coordonnées du repère image (x0, y0) point haut à gauche
    sans déformation de la courbe."""
    od = len(L)-1 # ordre de la courbe de Bézier
    assert od == 2 or od == 3, "Erreur le svg ne peut afficher l'ordre demandé"
    assert 2*mg < t , "Erreur la marge est trop grande  !"
    xMin, xMax, yMin, yMax = CadreBezier(L)
    a, b = xMax - xMin, yMax - yMin # dimensions du cadre origine
    if b == 0 or a/b > 1:
        rh = round((t - 2*mg)/a) # rapport homothétique (pixels par unité)
        ofx, ofy = x0, y0 + round(((t - 2*mg) - b*rh)/2)
    else:
        rh = round((t-2*mg)/b)
        ofx, ofy = x0 + round(((t-2*mg) - a*rh)/2), y0
    L1 = [(rh*(x - xMin) + ofx + mg, rh*(yMax - y) + ofy + mg) for (x,y) in L]
    if od == 2:
        tg= " Q" # Type Bézier Quadratique
    else:
        tg = " C" # Type Bézier Cubique
    chem = "M " + str(L1[0][0]) + "," + str(L1[0][1]) + tg
    for k in range(1, od+1):
        chem += " " + str(L1[k][0]) + "," + str(L1[k][1])
    return chem


def SvgCodeFigS4N(Lpar, t, x0=0, y0=0, mg=15, pr=0):
    """Générateur de code svg pour la représentation d'une figure présentant
    les 4 axes de symétrie d'un carré. Cette figure est générée par deux courbes
    de Bézier paramétrées par l'abscisse de A (point de départ de la première
    courbe situé sur l'axe de symétrie y=x), le point B ( point de départ de la
    seconde courbe ) et deux vecteurs u et v précisant les tangentes de la
    première courbe de Bézier en ces points. Lpar est une liste contenant
    [xA, xB , yB, xu, yu, xv, yv]. La seconde courbe est totalement
    contrainte par la première et les contraintes de symétrie. L'algorithme
    met la courbe à l'échelle pour l'intégrer un carré de côte t en préservant
    une marge mg avec l'un des bords sans déformation de la figure. On peut
    choisir une meilleure précision dans le calcul des pts de contrôle (par
    défaut arrondi à l'entier) en donnant à pr le nombre de décimales."""
    assert len(Lpar) == 7 and isinstance(Lpar[0],(int,float)) , "erreur de type de liste"
    c = 2*CadreBS4(Lpar) # coté de la croix en unité
    rh = (t - 2*mg)/c
    L = PFigS4ToListePts(Lpar)
    if pr > 0:
        pr = 10**pr
        Lpp = [(int(pr*(rh*(x+c/2)+x0+mg))/pr,int(pr*(rh*(c/2-y)+y0+mg))/pr) for (x,y) in L]
    else:
        Lpp = [(int(rh*(x+c/2)+x0+mg),int(rh*(c/2-y)+y0+mg)) for (x,y) in L]
    p1 = Lpp.pop(0)
    Chem = "M" + str(p1[0]) + "," + str(p1[1])
    for k in range(len(Lpp)):
        if k%7 == 0:
            tg = "\nC "
        elif k%7 == 3 or k%7 == 5:
            tg = " S"
        else:
            tg= " "
        Chem += tg + str(Lpp[k][0]) + "," + str(Lpp[k][1])
    return Chem
```