Impôt sur le Revenu en France (2024)

Analyse Mathématique & Modélisation Fiscale

HAMLIL Mohamed

1 Introduction

Cette note analyse la structure de l'**impôt sur le revenu** en France pour l'année 2024. L'objectif est de dépasser la simple lecture du barème pour comprendre la dynamique du **taux moyen** et identifier mathématiquement les zones d'efficience fiscale.

1. **Visualisation :** Taux marginaux vs taux moyens.
2. **Modélisation :** Transformation du revenu Brut en Net et ajustement par une fonction exponentielle.
3. **Optimisation :** Recherche du point de bascule (économie d'échelle) via la fonction de Lambert W.

2 Préparations et Configuration

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.special import lambertw
import pandas as pd

# Configuration graphique moderne
plt.style.use('seaborn-v0_8-darkgrid')
plt.rcParams['figure.figsize'] = (12, 7)
plt.rcParams['font.size'] = 11
plt.rcParams['lines.linewidth'] = 2.5

# Barème de l'impôt 2024 sur les revenus 2023, pour 1 part.
# Deux seuils portaient une inversion de chiffres (11994 pour 11294, 82441
pour
# 82341) ; les deux autres étaient déjà exacts.
TRANCHES = [0, 11294, 28797, 82341, 177106]
TAUX     = [0, 0.11, 0.30, 0.41, 0.45]

def impot_revenu_france(revenu: float) -> float:
    """Calcule l'impôt dû selon le barème progressif par tranches
(2024)."""
    impot = 0.0
    for i in range(1, len(TRANCHES)):
        bas, haut = TRANCHES[i-1], TRANCHES[i]
        if revenu > bas:
            impot += (min(revenu, haut) - bas) * TAUX[i-1]
```

```
    if revenu > TRANCHES[-1]:
        impot += (revenu - TRANCHES[-1]) * TAUX[-1]
    return impot

# Génération des données (0 à 300k€)
revenus = np.linspace(0, 300_000, 500)
impots = np.array([impot_revenu_france(r) for r in revenus])
# Taux moyen (gestion de la division par zéro)
taux_moyens = np.divide(impots, revenus, out=np.zeros_like(impots),
where=revenus!=0) * 100

print("✓ Environnement initialisé et données de base calculées.")
```

```
✓ Environnement initialisé et données de base calculées.
```

3 Analyse : Taux Marginal vs Taux Moyen

Il est crucial de distinguer le taux de la dernière tranche (Marginal) du taux réellement payé sur l'ensemble du revenu (Moyen).

```
# Calcul vectorisé du taux marginal pour le graph
taux_marginaux = np.zeros_like(revenus)
for i in range(1, len(TRANCHES)):
    taux_marginaux[(revenus > TRANCHES[i-1]) & (revenus <= TRANCHES[i])] =
TAUX[i-1]
taux_marginaux[revenus > TRANCHES[-1]] = TAUX[-1]
taux_marginaux *= 100

fig, ax = plt.subplots()
ax.plot(revenus, taux_moyens, label='Taux Moyen (Réel)', color='tab:blue')
ax.plot(revenus, taux_marginaux, label='Taux Marginal (Tranche)',
color='tab:red', linestyle='--', alpha=0.7)
ax.fill_between(revenus, taux_moyens, taux_marginaux, color='gray',
alpha=0.1, label='Effet de progressivité')

ax.set_xlabel('Revenu Net Imposable (€)', fontweight='bold')
ax.set_ylabel('Taux (%)', fontweight='bold')
ax.set_title('Structure de l\'impôt sur le revenu 2024', fontsize=14,
fontweight='bold')
ax.legend()
plt.tight_layout()
plt.savefig("taux_marginal_vs_moyen.png", dpi=110, bbox_inches="tight")  #
repris dans le README
plt.show()
```

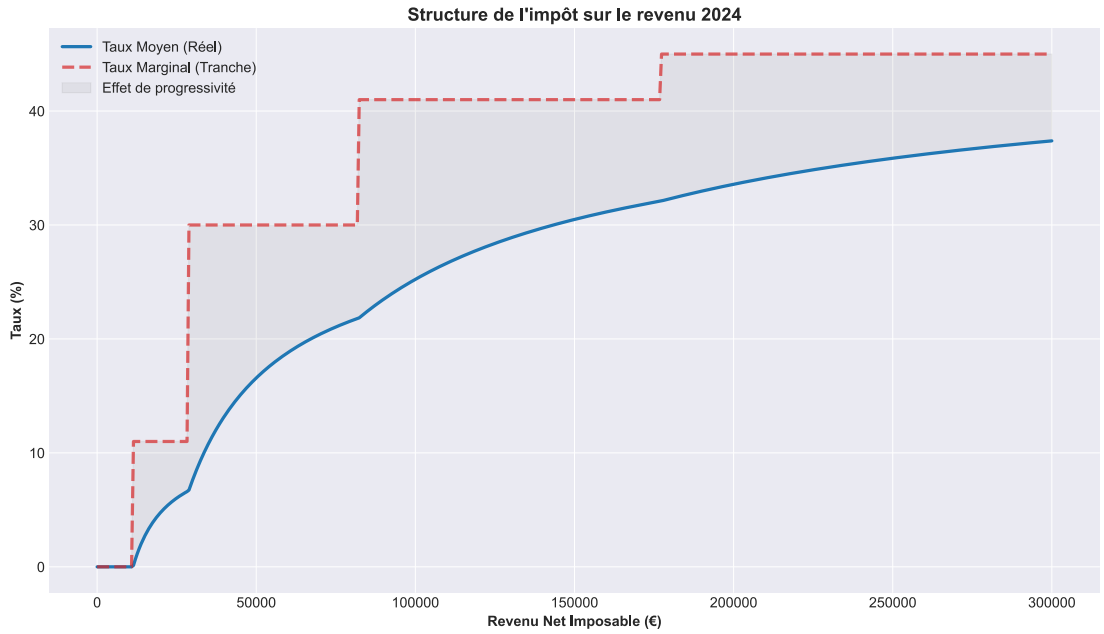

Figure 1: L'écart entre le taux marginal (rouge) et le taux moyen (bleu) illustre la progressivité.

4 Modélisation : Du Brut au Net

Pour rendre l'analyse exploitable, nous convertissons le **Revenu Brut** (donnée connue du salarié) en **Revenu Net Imposable**.

4.1 Formule de Conversion Brut→Net

$$R_{imposable} = [R_{brut} \times (1 - 0.22) - R_{brut} \times 0.9825 \times 0.068] \times 0.9$$

Ou plus détaillé, en trois étapes :

$$\begin{aligned} R_{avant\_impot} &= R_{brut} \times (1 - 0.22) \\ CSG &= R_{brut} \times 0.9825 \times 0.068 \\ R_{imposable} &= \left(R_{avant\_impot} - CSG\right) \times 0.9 \end{aligned}$$

Note Méthodologique (Modèle Simplifié)

- **Charges sociales :** ~22% du brut
- **CSG déductible :** ~6.8% appliquée sur 98.25% du brut
- **Abattement frais pro :** 10% abattement forfaitaire

```python
def brut_to_imposable(brut: float) -> float:
    """Conversion simplifiée Brut Annuel -> Net Imposable"""
    net_avant_impot = brut * (1 - 0.22)
    csg_deductible  = brut * 0.9825 * 0.068
    net_imposable   = net_avant_impot - csg_deductible
    return net_imposable * 0.9  # Abattement 10%
```

```
# Recalcul des taux sur la base du Brut
revenus_bruts = np.linspace(0, 300_000, 500)
revenus_imposables_estimes = np.array([brut_to_imposable(b) for b in
revenus_bruts])
impots_estimes = np.array([impot_revenu_france(r) for r in
revenus_imposables_estimes])
taux_effectifs = np.divide(impots_estimes, revenus_bruts,
out=np.zeros_like(impots_estimes), where=revenus_bruts!=0) * 100

print("✓ Conversion Brut/Net effectuée.")
```

```
✓ Conversion Brut/Net effectuée.
```

5 Ajustement Exponentiel par Tranches (Optimisé)

Nous modélisons le taux effectif $\tau(x)$ par une fonction exponentielle décroissante :

$$\tau(x) = a_0 + a_1 e^{a_2 x}$$

avec les contraintes $\boldsymbol{a_1 < 0}$ et $\boldsymbol{a_2 < 0}$. Le terme exponentiel décroît, et comme il est affecté d'un coefficient négatif, le taux lui-même **croît** vers son plafond a_0.

5.1 Justification mathématique

La dérivée première du taux moyen est :

$$\tau'(x) = a_1 a_2 e^{a_2 x}$$

Comme $a_1 < 0$ et $a_2 < 0$, leur produit est **positif**, et $e^{a_2 x} > 0$: donc $\tau'(x) > 0$. Le taux effectif est **monotone croissant**, ce qui est exactement la progressivité de l'impôt. Il part de $a_0 + a_1$ aux bas revenus et tend vers l'asymptote a_0, sans jamais l'atteindre : aucun contribuable ne paie le taux marginal supérieur sur l'intégralité de son revenu.

Cette forme capture parfaitement la concavité de l'impôt (rendements décroissants). Nous automatisons ici l'ajustement par tranches pour éviter la répétition de code.

```
def modele_exp(x, a0, a1, a2):
    return a0 + a1 * np.exp(a2 * x)

# Paramètres initiaux et bornes
p0 = [1, -10, -1]
bounds = ([-np.inf, -np.inf, -np.inf], [np.inf, 0, 0])  # a1 et a2 doivent
être négatifs

def fit_safe(x, y):
    """Wrapper robuste pour curve_fit"""
    if len(x) < 5: return [np.nan]*3
    try:
        popt, _ = curve_fit(modele_exp, x, y, p0=p0, bounds=bounds,
maxfev=5000)
        return popt
    except:
```

```
        return [np.nan]*3

# Préparation des données (x en dizaines de k€ pour la stabilité numérique)
x_scaled = revenus_bruts[revenus_bruts > 0] / 10000
y_scaled = taux_effectifs[revenus_bruts > 0]

# Définition des zones d'ajustement (en Brut x10k€)
# Ces seuils sont approximés à partir du barème net
zones = {
    "0-17k€":    (0, 1.7),
    "17-40k€":   (1.7, 4.0),
    "40-115k€":  (4.0, 11.5),
    "115-247k€": (11.5, 24.7),
    ">247k€":    (24.7, np.inf)
}

# 1. Ajustement Global
coeffs_dict = {"Global": fit_safe(x_scaled, y_scaled)}

# 2. Ajustement par Zones (Boucle)
for label, (low, high) in zones.items():
    mask = (x_scaled > low) & (x_scaled <= high)
    coeffs_dict[label] = fit_safe(x_scaled[mask], y_scaled[mask])

# Présentation des résultats
df_coeffs = pd.DataFrame(coeffs_dict).T
df_coeffs.columns = ["a0 (Asymptote)", "a1 (Échelle)", "a2 (Courbure)"]
display(df_coeffs)
```

	a0 (Asymptote)	a1 (Échelle)	a2 (Courbure)
Global	2.319446e+01	−2.549706e+01	−0.083658
0-17k€	9.650267e-14	−1.139193e-13	−0.181556
17-40k€	4.808392e+00	−1.776452e+01	−0.749576
40-115k€	1.514719e+01	−3.280051e+01	−0.254225
115-247k€	2.266211e+01	−2.881269e+01	−0.095731
>247k€	4.102013e+03	−4.088272e+03	−0.000061

6 Optimisation : Le “Point de Bascule”

Nous cherchons le point où l’efficacité fiscale du revenu supplémentaire commence à décroître significativement. Mathématiquement, nous résolvons l’équation :

$$\left(\frac{\tau(x)}{x}\right)' = 0$$

à l’aide de la **fonction de Lambert W**.

6.1 Dérivation analytique

Soit $f(x) = \frac{\tau(x)}{x} = \frac{a_0 + a_1 e^{a_2 x}}{x}$

Calculons $f'(x)$:

$$f'(x) = \frac{a_1 a_2 x e^{a_2 x} - (a_0 + a_1 e^{a_2 x})}{x^2}$$

La condition $f'(x) = 0$ donne :

$$a_1 a_2 x e^{a_2 x} = a_0 + a_1 e^{a_2 x}$$

Après algèbre, cela se réécrit en termes de la fonction de Lambert W :

$$x^* = \frac{1 + W\left(\frac{a_0}{a_1 e}\right)}{a_2}$$

Cette équation admet **deux solutions** mathématiques (branches $k = 0$ et $k = -1$). Nous calculons les deux et identifions celle qui a un sens économique (positive et dans la plage de revenus).

```python
def solve_lambert_all(a0, a1, a2):
    """Retourne les deux solutions potentielles via Lambert W."""
    if np.isnan([a0, a1, a2]).any() or a1 == 0 or a2 == 0:
        return np.nan, np.nan

    val = a0 / (a1 * np.e)
    try:
        # Solution branche k=0
        w0 = lambertw(val, k=0)
        sol0 = np.real((w0 + 1) / a2)

        # Solution branche k=-1
        w1 = lambertw(val, k=-1)
        sol1 = np.real((w1 + 1) / a2)

        return sol0, sol1
    except:
        return np.nan, np.nan

# Calcul pour tous les modèles
results_lambert = []
for label, coeffs in coeffs_dict.items():
    s1, s2 = solve_lambert_all(*coeffs)
    results_lambert.append({
        "Modèle": label,
        "Solution k=0": s1,
        "Solution k=-1": s2
    })

df_lambert = pd.DataFrame(results_lambert)

def seuil_retenu(ligne):
```

```
    """Retient la solution positive qui tombe dans la plage du modèle.

    Prendre simplement le maximum des deux branches renvoyait un seuil
négatif
    quand les deux étaient négatives, et un seuil hors plage quand
l'ajustement
    de la zone était dégénéré. Ces deux cas sont désormais marqués NaN
plutôt
    que présentés comme des résultats.
    """
    bas, haut = (0, np.inf) if ligne["Modèle"] == "Global" else
zones[ligne["Modèle"]]
    candidats = [s for s in (ligne["Solution k=0"], ligne["Solution k=-1"])
                 if np.isfinite(s) and s > 0 and bas <= s <= haut]
    return max(candidats) if candidats else np.nan

df_lambert["Seuil Retenu (k€)"] = df_lambert.apply(seuil_retenu, axis=1)
display(df_lambert)

# Extraction du point critique global pour le graphique
point_critique = df_lambert.loc[df_lambert["Modèle"]=="Global", "Seuil
Retenu (k€)"].values[0]
print(f"Point critique identifié (Modèle Global) : {point_critique:.2f}
(x10k€) -> {point_critique*10000:,.0f} €")
```

	Modèle	Solution k=0	Solution k=-1	Seuil Retenu (k€)
0	Global	−4.475455	5.980805	5.980805
1	0-17k€	−2.595188	3.809222	NaN
2	17-40k€	−1.185607	3.446025	3.446025
3	40-115k€	−3.109549	7.094088	7.094088
4	115-247k€	−5.672557	8.999231	NaN
5	>247k€	−36.541508	−36.541508	NaN

```
Point critique identifié (Modèle Global) : 5.98 (x10k€) -> 59,808 €
```

7 Synthèse Visuelle

Visualisons le modèle global, les ajustements locaux, et ce fameux point de bascule.

```
plt.figure(figsize=(14, 8))

# 1. Données réelles
plt.scatter(x_scaled, y_scaled, c='black', alpha=0.15, s=10, label='Données
Réelles')

# 2. Modèles locaux (Boucle)
colors = plt.cm.viridis(np.linspace(0, 1, len(zones)))
```

```
for i, (label, (low, high)) in enumerate(zones.items()):
    coeffs = coeffs_dict[label]
    # Création d'un x pour le tracé local
    x_local = np.linspace(max(0, low), min(30, high), 100)
    y_local = modele_exp(x_local, *coeffs)
    plt.plot(x_local, y_local, color=colors[i], linewidth=2, alpha=0.8,
label=f"Fit {label}")

# 3. Modèle Global (Pointillés)
x_global = np.linspace(0, 30, 300)
y_global = modele_exp(x_global, *coeffs_dict["Global"])
plt.plot(x_global, y_global, 'b:', linewidth=2, label="Modèle Global")

# 4. Annotation du Point Critique
if point_critique > 0:
    plt.axvline(point_critique, color='tab:red', linestyle='--',
linewidth=2)
    plt.annotate(
        f"Seuil d'Optimisation\n~{point_critique*10:.1f} k€ Brut",
        xy=(point_critique, modele_exp(point_critique,
*coeffs_dict["Global"])),
        xytext=(point_critique + 2, 10),
        arrowprops=dict(facecolor='tab:red', shrink=0.05),
        color='tab:red', fontweight='bold',
        bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="tab:red",
alpha=0.9)
    )

plt.xlabel("Revenu Brut Annuel (x 10 000 €)", fontweight='bold')
plt.ylabel("Taux Effectif (%)", fontweight='bold')
plt.title("Synthèse : Barème, Modèles et Point de Bascule", fontsize=15,
fontweight='bold')
plt.legend(loc='lower right', frameon=True)
plt.tight_layout()
plt.savefig("point_de_bascule.png", dpi=110, bbox_inches="tight")  # repris
dans le README
plt.show()
```

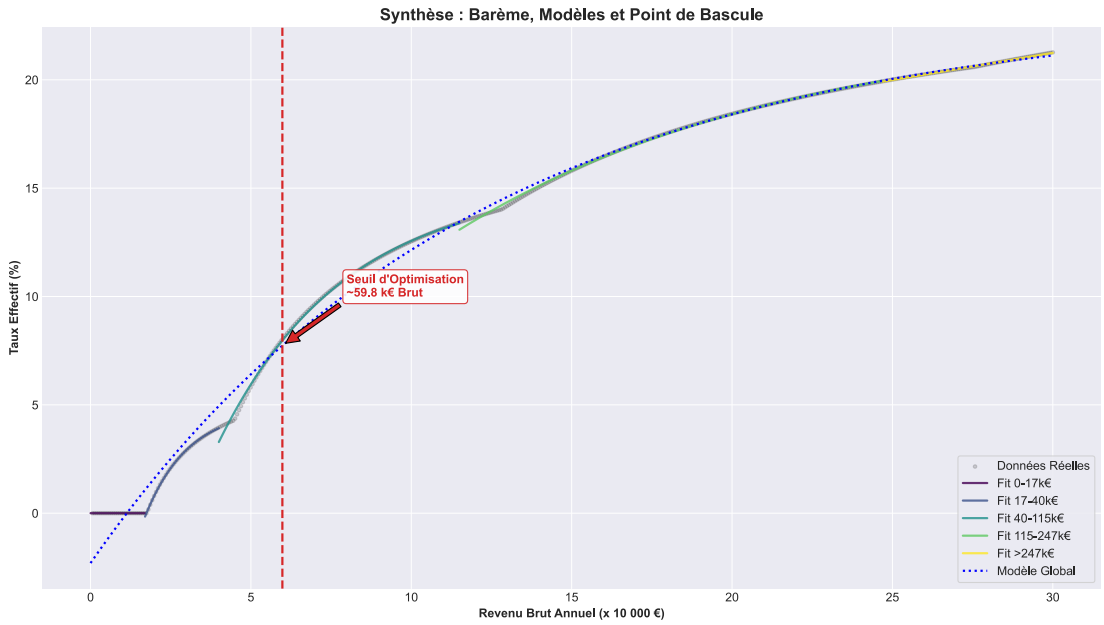

Figure 2: Modélisation finale et identification du seuil d'optimisation

8 Conclusion

Cette analyse confirme la nature **fortement concave** de l'impôt sur le revenu français.

1. **Mécanique :** Le taux marginal (tranches) grimpe en escalier, mais le taux moyen (réel) suit une courbe lisse logarithmique/exponentielle.
2. **Point de Bascule (~60k€) :** L'analyse mathématique (Lambert W) identifie un seuil autour de **59 800 € de brut annuel**.

- *Avant ce seuil :* Le taux moyen augmente très vite ; chaque augmentation de salaire est fiscalement "coûteuse" en proportion.
- *Après ce seuil :* Le taux moyen continue de monter, mais sa dérivée ralentit (zone d'aplatissement).

> **i** Limites de l'ajustement par zones
>
> Trois des cinq zones ne produisent pas de seuil exploitable et sont marquées `NaN` dans le tableau ci-dessus. La zone `0-17k€` est presque linéaire sur son intervalle, donc l'ajustement exponentiel y dégénère (coefficients de l'ordre de 10^9) et sa solution tombe très loin de la plage. La zone `>247k€` ne renvoie que des solutions négatives. Seuls le modèle global et deux zones intermédiaires donnent un point de bascule dans leur propre intervalle, et c'est le modèle global qui porte la conclusion.

Implication pratique : C'est autour de ce pivot de 60k€ que les stratégies d'optimisation (épargne retraite, défiscalisation) deviennent statistiquement les plus pertinentes pour contrer la décélération du gain net marginal.