Université de Paris

Projet de calcul scientifique
Résolution d'équations non linéaires

Terai Karim, Hivart Corentin

Mai 2023

Algorithme de la dichotomie

- $x_{k+1} = x_k, \quad y_{k+1} = z_k, \quad \text{si } f(z_k) > 0$
- $x_{k+1} = z_k, \quad y_{k+1} = y_k, \quad \text{si } f(z_k) < 0$
- $x_{k+1} = y_{k+1} \quad \text{si } f(z_k) = 0$
- $z_{k+1} = \dfrac{x_k + y_k}{2}$

Convergence

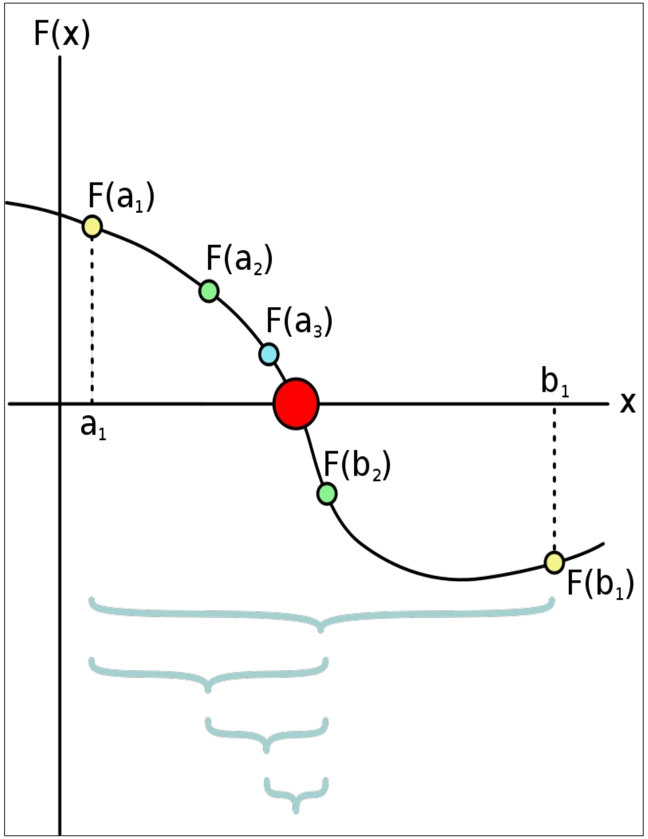
Soit $f \in \mathcal{C}([a,b],\mathbb{R})$ *tel que* $f(a)f(b) < 0$.

La méthode de dichotomie converge linéairement à taux $\alpha = \dfrac{1}{2}$

$$\|x_k - x_{sol}\| \leq C\left(\frac{1}{2}\right)^k$$

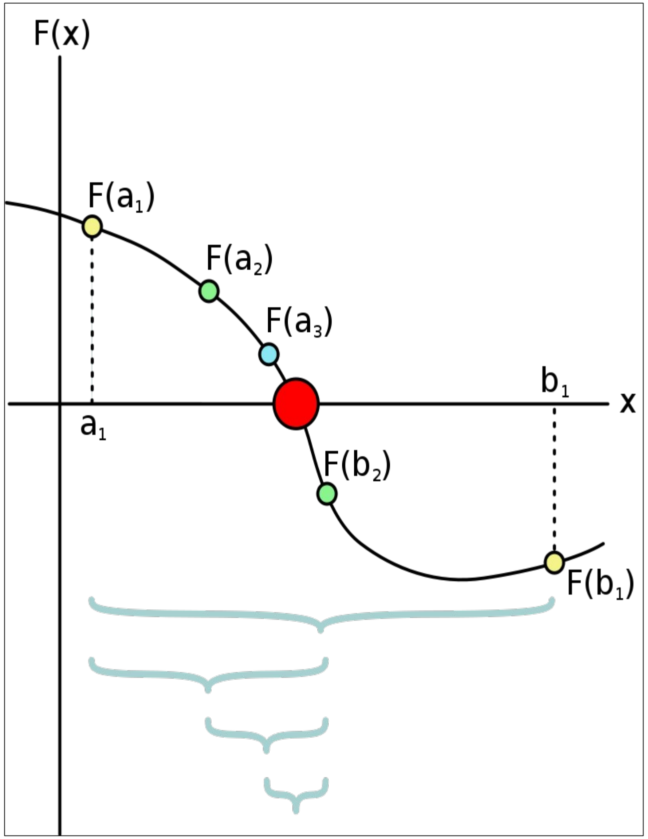
Contraintes

- $f \in \mathcal{C}([a,b],\mathbb{R})$
- f doit admettre une racine entre a et b
- $f(a)f(b) < 0.$



Algorithme de la dichotomie

```
def dichotomie(f,a,b,precision=1e-6,n=200):
    fa = f(a)
    fb = f(b)
    compteur=0
    assert fa*fb <= 0
    assert a<b
    if fa==0:
        return a
    if fb==0:
        return b
    while b-a > 2*precision and compteur<n:
        m = (a+b)/2
        fm = f(m)
        compteur += 1
        if fa*fm<=0:
            b, fb = m, fm
        else:
            a, fa = m, fm
    print(compteur)
    return (a+b)/2
```



Algorithme de Newton

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Convergence

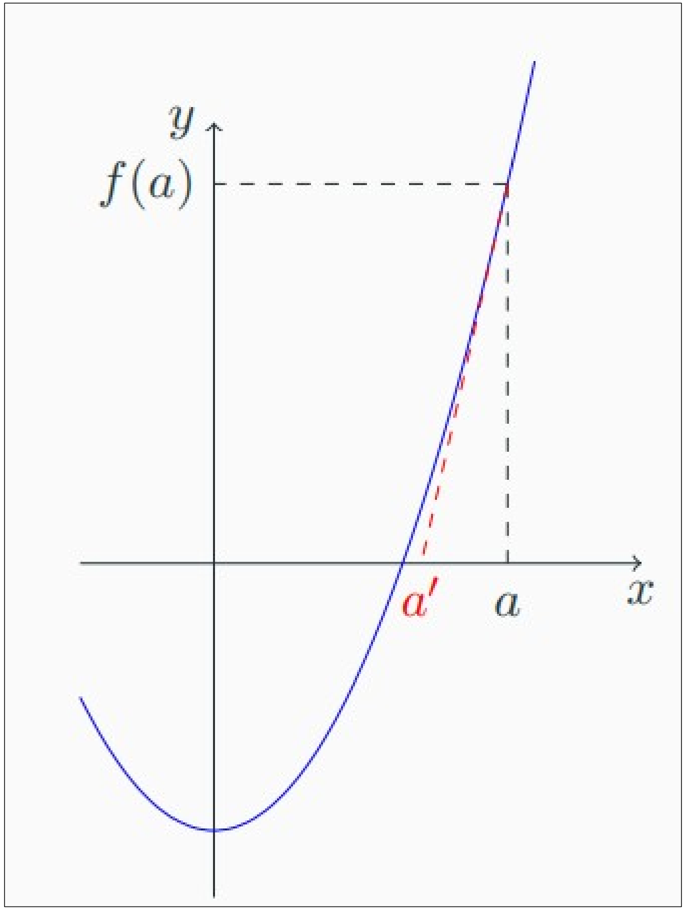
Soit $f \in \mathcal{C}^2(\mathbb{R};\mathbb{R})$ *et* $x^* \in \mathbb{R}$ *tel que :*

$$f(x^*) = 0; \quad f'(x^*) \neq 0$$

Alors il existe $\epsilon > 0$ *tel que pour toute initialisation* $x_0 \in]x^* - \epsilon; x^* + \epsilon[$, *la méthode de Newton initialisée en* x_0 *converge quadratiquement vers* x^*

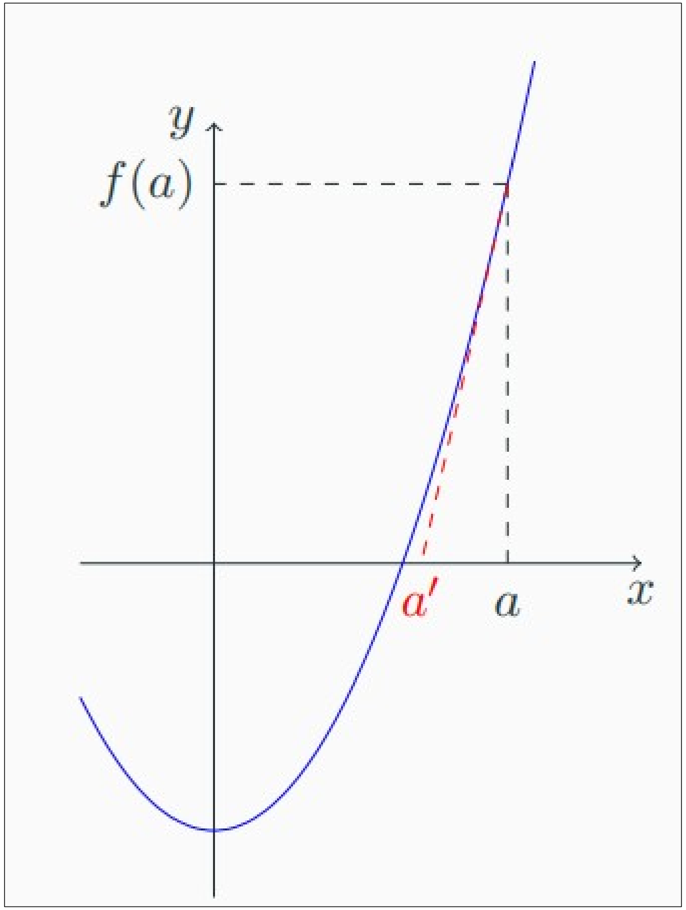
Contraintes

- $f \in \mathcal{C}^1(\mathbb{R};\mathbb{R})$
- Initialiser près de la solution



Algorithme de Newton

```
def NewtonR1(f,g,x0,precision=1e-6,n=200):
    x = x0
    compteur=0
    while (compteur<n and np.abs(f(x)) > precision):
        compteur += 1
        assert g(x) != 0
        x = x - (f(x)/g(x))
    if np.abs(f(x)) > precision:
        print("L'algorithme ne converge pas en "+str(n)+" etapes")
        return False
    print(compteur)
    return x
```



Algorithme de la sécante

$$x_{k+1} = x_k - (x_k - x_{k-1})\frac{f(x_k)}{f(x_k) - f(x_{k-1})}$$

Convergence

Soit $f \in \mathcal{C}^2(\mathbb{R};\mathbb{R})$ *et* $x^* \in \mathbb{R}$ *tel que :*

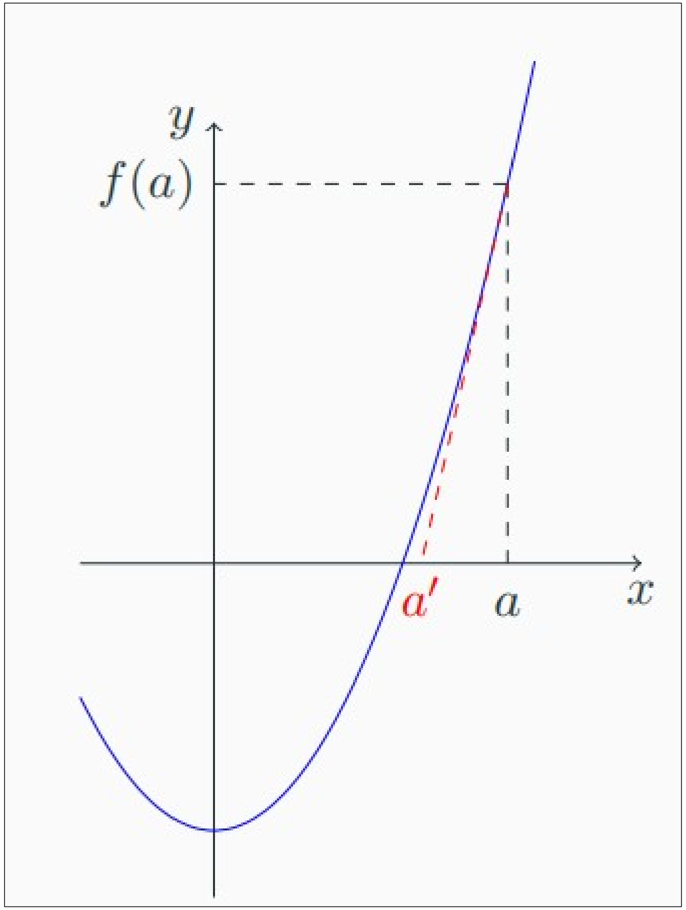
$$f(x^*) = 0; \quad f'(x^*) \neq 0$$

Si x_0 *et* x_1 *sont suffisamment proches de* x^**, alors la suite* $(x_k)_{k\in\mathbb{N}}$ *définie par la méthode de la sécante converge vers* x^* *à l'ordre au moins*

$$\phi = \frac{1}{2}(1+\sqrt{5}) \approx 1.618$$

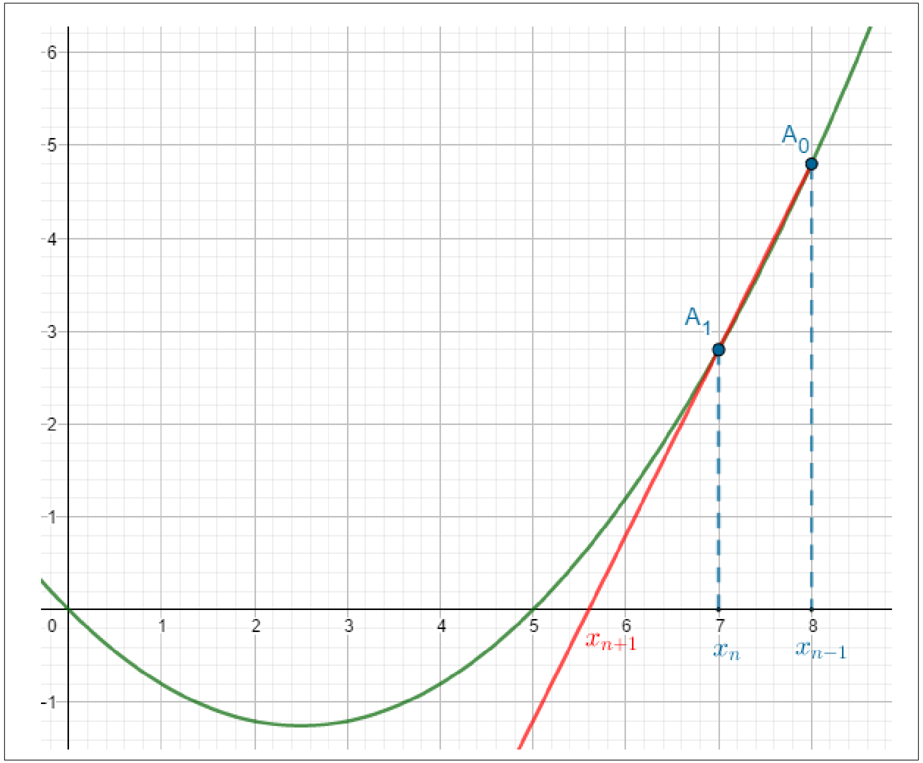
Contraintes

- $f \in \mathcal{C}(\mathbb{R};\mathbb{R})$
- Initialiser près de la solution



Algorithme de la sécante

```
def secante(f, x0, x1, precision=1e-6, n=200):
    x_prec, x_act = x0, x1
    compteur = 0
    while compteur < n and np.abs(f(x_act)) > precision:
        f_prec = f(x_prec)
        f_act = f(x_act)
        assert((f_act - f_prec)!=0)
        x_nouv = x_act - (x_act - x_prec) * (f_act / (f_act - f_prec))
        x_act, x_prec = x_nouv, x_act
        compteur += 1
    if np.abs(f(x_act)) > precision:
        print("L'algorithme ne converge pas en " + str(n) +" etapes")
        return False
    print(compteur)
    return x_act
```



Algorithme de Newton-Raphson en dimension n

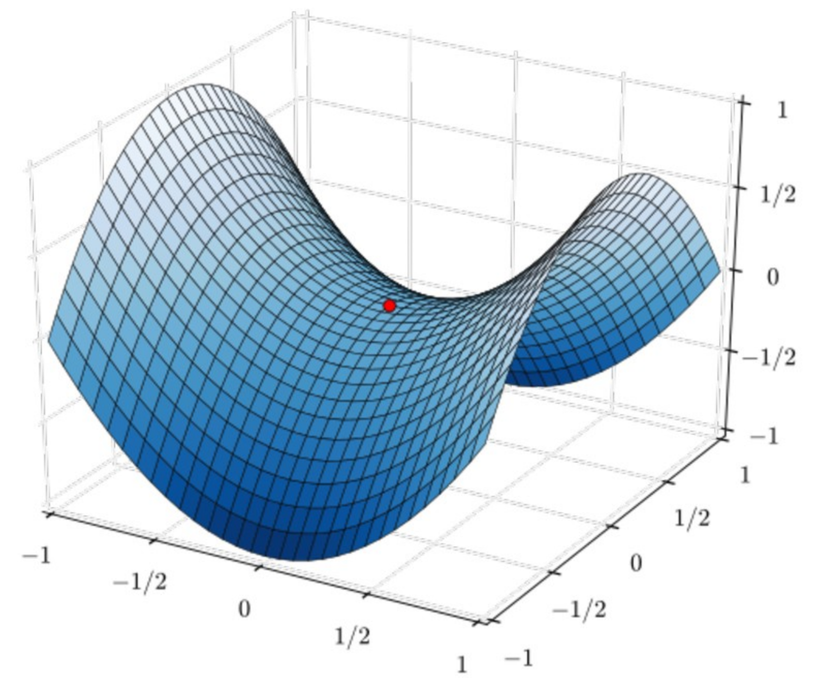
$$Jac_F(x_k)(x_{k+1} - x_k) = -F(x_k)$$

$$x_{k+1} = x_k - (Jac_F)^{-1}(x_k)F(x_k)$$

Convergence et Contraintes

- Dérivable sur $\mathbb{R}$ $\Longleftrightarrow$ Différentiable sur $\mathbb{R}^n$
  Dérivée sur $\mathbb{R}$ — Jacobienne sur $\mathbb{R}^n$
  $f(x) = 0$ — $||F(x)|| = 0$

- Sinon, nous avons les même convergence et contraintes que l'algorithme de Newton avec des points de n coordonnées.

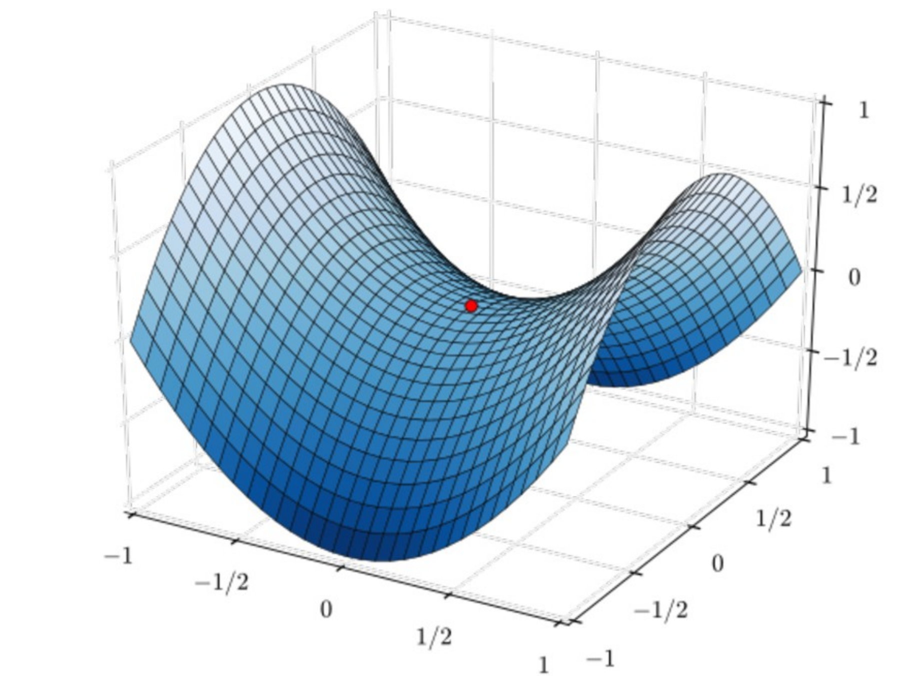


Représentation graphique de f(x,y) = x² - y²

Algorithme de Newton-Raphson en dimension n

```
def approx_jacob(F, X, h=1e-6):
    n = len(X)
    J = np.zeros((n, n))
    for i in range(n):
        dX = np.zeros(n)
        dX[i] = h

        F_plus = F(X + dX)
        F_moins = F(X - dX)

        J[:, i] = (F_plus - F_moins) / (2 * h)

    return J
```

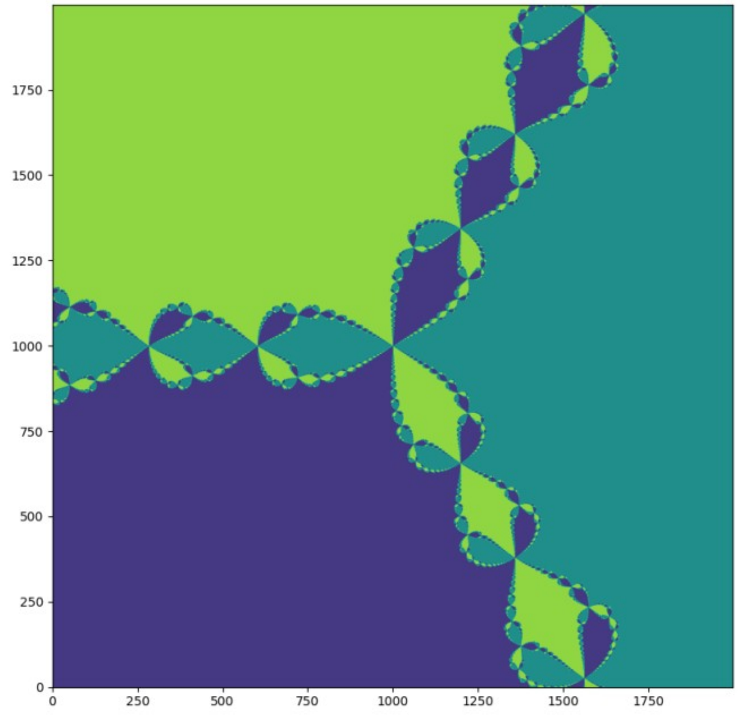
```
def NewtonRn(F,X0, precision=1e-6, n=200):
    X = X0
    compteur = 0
    norme = np.linalg.norm(F(X))
    while norme > precision and compteur < n:
        jac = approx_jacob(F,X)
        delta = np.linalg.solve(jac, -F(X))
        X += delta
        compteur += 1
        norme = np.linalg.norm(F(X))
    if norme > precision:
        print("L'algorithme ne converge pas en " + str(n) +" etapes")
        return False
    print(compteur)
    return X
```



Représentation graphique de $f(x,y) = x^2 - y^2$

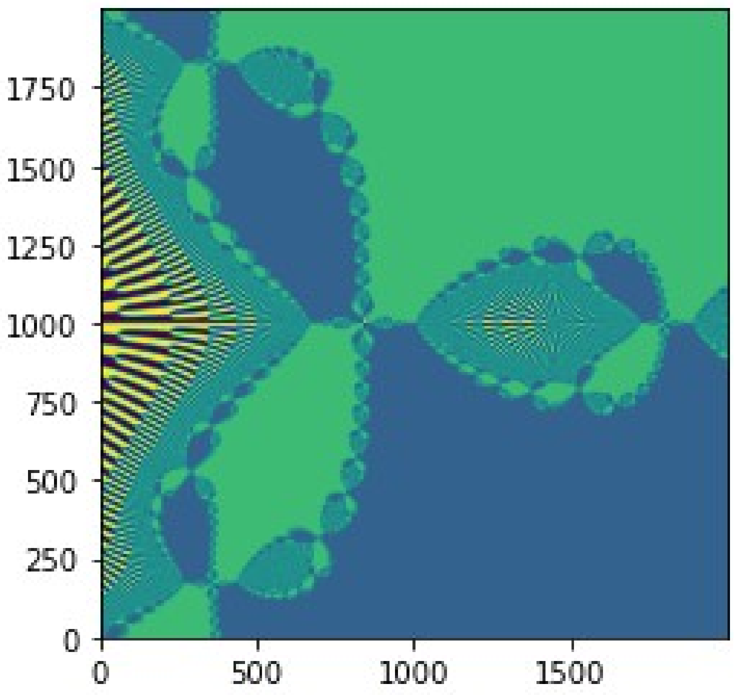
Algorithme de la fractale de Newton

```
def df(f, x, h=1e-6):
    return((f(x+h)-f(x-h))/(2*h))


def newton(z,f):
    for i in range(20):
        z = z - f(z) / df(f,z)
    return z


def fractale(g,resolution=500):
    X = np.linspace(-2, 2, resolution)
    Y = np.linspace(-2, 2, resolution)
    x, y = np.meshgrid(X, Y)
    z = x + 1j*y # On utilise la correlation R2 —— C

    w = np.zeros_like(z)
    for i in range(len(z)):
        for j in range(len(z[0])):
            w[i][j] = newton(z[i][j],g)
    plt.imshow(np.angle(w))
    plt.gca().invert_yaxis()
    plt.show()
```

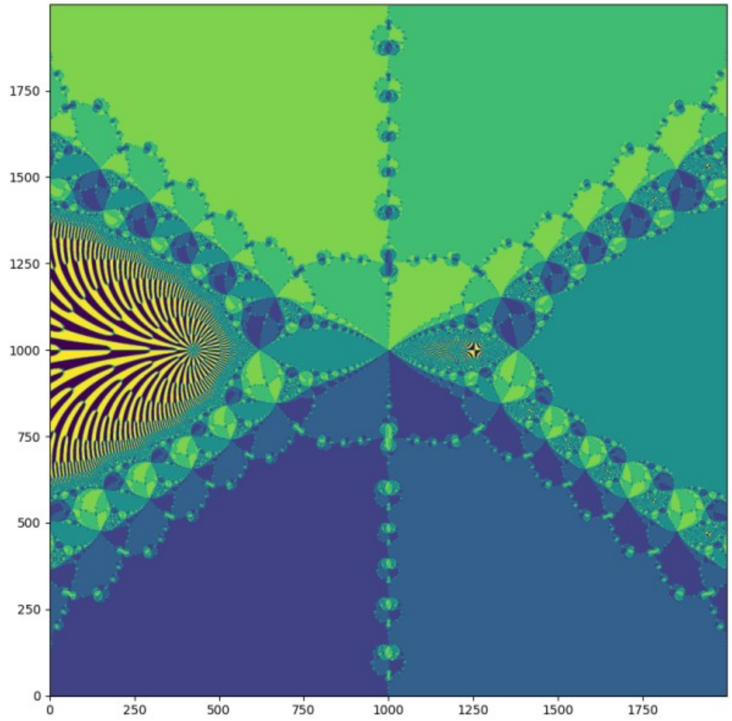


Fractale de Newton du polynôme $p(z) = z^3 - 1$

Algorithme de la fractale de Newton: Exemples inexpliqués



Fractale de Newton insatisfaisante du polynôme $p(z) = z^3 + z^2 + 14$



Fractale de Newton insatisfaisante du polynôme $p(z) = -z^6 - z^2 - 1$

Université de Paris

Conclusion

Fonction	Dichotomie	Sécante 1	Sécante 2	Newton 1	Newton 2	Newton 3
f_1	24	11	23	6	18	23
f_2	24	8	Non CV	4	153	Non CV
f_3	22	5	5	2	2	4
f_4	22	7	Erreur	4	Erreur	Erreur
f_5	23	5	14	4	10	13