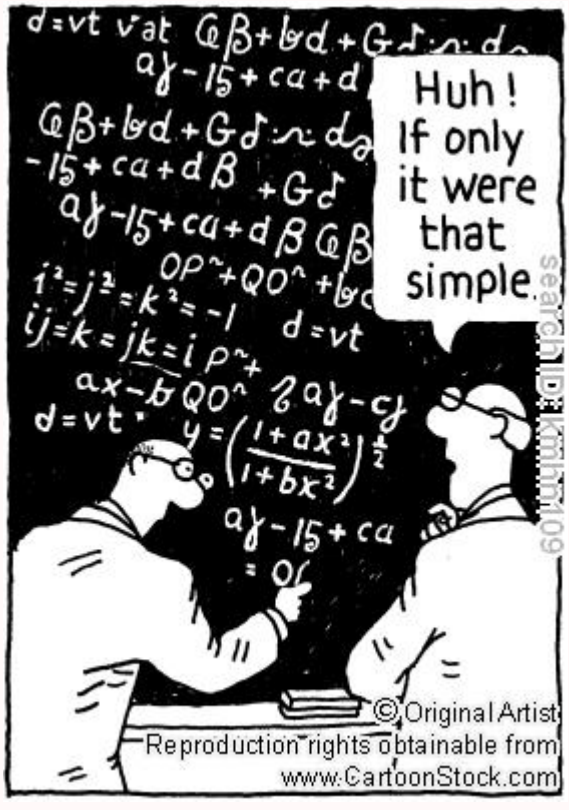
ANNEXE

# F

# Pour en finir avec les résolutions numériques d'équations différentielles



## Position du problème

Soit $U$ un ouvert de $\mathbb{R}\times\mathbb{R}^n$ et $f:U\longrightarrow\mathbb{R}^n$ une application continue sur l'ouvert $U$.
Alors pour $(t_0,x_0)\in U$, on appelle solution du problème de Cauchy :

$$\begin{cases} x'(t)=f(t,x(t)) \\ x(t_0)=x_0 \end{cases} \qquad (C)$$

tout couple $(I,x)$ où $I$ est un intervalle contenant $t_0$ et $x:I\longrightarrow\mathbb{R}^n$ une solution sur $I$ de l'équation différentielle $x'(t)=f(t,x(t))$ telle que $x(t_0)=x_0$.
L'équation différentielle qui apparaît dans $(C)$ sera appelée équation différentielle scalaire si $n=1$, et système différentiel sinon.
Si l'application $f:U\longrightarrow\mathbb{R}^n$ est de classe $C^1$ sur l'ouvert $U$, alors, d'après le théorème de Cauchy-Lipschitz, pour $(t_0,x_0)\in U$, il existe une unique solution maximale au problème de Cauchy $(C)$.
Dans la plupart des cas, on ne sait pas résoudre explicitement le problème de Cauchy $(C)$ ; d'où la nécessité de mettre au point des méthodes numériques de résolution approchée d'un tel problème.

## Méthode d'Euler

La méthode d'Euler est la méthode la plus simple pour résoudre numériquement une équation différentielle. Elle présente un réel intérêt théorique, puisque elle peut être utilisée pour démontrer le théorème de Cauchy-Lipschitz. Toutefois, son intérêt pratique est limité par sa faible précision.
L'idée d'Euler consiste à utiliser l'approximation :

$$x'(t)\approx\frac{x(t+h)-x(t)}{h} \qquad \text{pour } h \text{ petit}$$

Autrement dit,

$$x(t+h)\approx x(t)+h\cdot x'(t)=x(t)+h\cdot f(t,x(t))$$

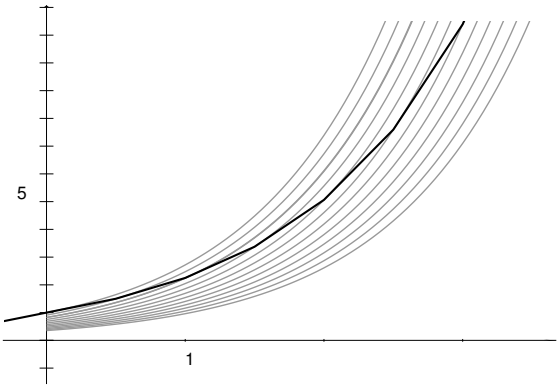
Partant du point $(t_0,x_0)$, on suit alors la droite de pente $f(t_0,x_0)$ sur l'intervalle de temps $t_0t_0+h$.
On pose alors :

$$\begin{cases} t_1=t_0+h \\ x_1=x_0+h\cdot f(t_0,x_0) \end{cases}$$

De nouveau, partant du point $(t_1,x_1)$, on suit alors la droite de pente $f(t_1,x_1)$ sur l'intervalle de temps $t_1t_1+h$, et ainsi de suite.
On construit ainsi une suite de points de la manière suivante :

$$\forall k\in[\![0,N]\!],\quad \begin{cases} t_{k+1}=t_k+h \\ x_{k+1}=x_k+h\cdot f(t_k,x_k) \end{cases}$$

La ligne brisée joignant les points $(t_k,x_k)k\in[\![0,N]\!]$ donnera une solution approchée de notre équation différentielle.
Il faut bien comprendre qu'à chaque étape, on repart dans la direction d'une solution exacte de l'équation différentielle, mais qui n'est pas celle qui est solution du problème de Cauchy initial. Sur le graphique ci-contre, on trace une solution approchée pour la condition initiale $x(0)=1$ et les courbes intégrales de notre équation qui passent par les points $(t_k,x_k)$.




Pour juger de la qualité d'une méthode (ou schéma) numérique de résolution d'équations différentielles, il faut prendre en compte plusieurs critères.

— L'erreur de consistance donne l'ordre de grandeur de l'erreur effectué à chaque pas. Par exemple, dans le cas de la méthode d'Euler, l'erreur de consistance mesure l'erreur qu'entraîne le fait d'approcher le nombre dérivé par un taux d'accroissement. La sommation des erreurs de consistance à chaque pas donnera l'ordre de grandeur de l'erreur globale (sous des hypothèses de régularité pour la fonction $f$). À l'aide de la formule de Taylor-Lagrange, on peut montrer que dans le cas de la méthode d'Euler, l'erreur de consistance est dominée par $h^2$ ; on dit alors que cette méthode est d'ordre 1. Ainsi, d'un point de vue théorique, plus le pas est petit, meilleure sera l'approximation.

Un schéma numérique est dit consistant si la somme des erreurs de consistance tend vers 0 quand le pas $h$ tend vers 0 ; ce qui est le cas de la méthode d'Euler (en gros, car $Nh^2 \sim h \to 0$ où $N$ désigne le nombre d'itérations).

— La stabilité contrôle la différence entre deux solutions approchées correspondant à deux conditions initiales voisines : un schéma est dit stable si un petit écart entre les conditions initiales $x(t_0) = x_0$ et $\tilde{x}(t_0) = x_0 + \varepsilon$ et de petites erreurs d'arrondi dans le calcul récurrent des $\tilde{x}_k$ provoquent une erreur finale $x_k - \tilde{x}_k$ contrôlable. La méthode d'Euler est une méthode stable.

— Un schéma est dit convergent lorsque l'erreur globale (qui est le maximum des écarts entre la solution exacte et la solution approchée) tend vers 0 lorsque le pas tend vers 0. En fait, pour que le schéma soit convergent, il suffit que la méthode soit consistante et stable.

— Enfin pour la mise en application du schéma, il faut aussi prendre en compte l'influence des erreurs d'arrondi. En effet, afin de minimiser l'erreur globale théorique, on pourrait être tenté d'appliquer la méthode d'Euler avec un pas très petit, par exemple de l'ordre de $10^{-16}$, mais ce faisant, outre que le temps de calcul deviendrait irréaliste, les erreurs d'arrondi feraient diverger la solution approchée très rapidement, puisque pour des flottants de l'ordre de $10^{-16}$, les calculs ne sont plus du tout exacts !

En pratique, il faut prendre $h$ assez petit (pour que la méthode converge assez rapidement du point de vue théorique), mais pas trop petit non plus (pour que les erreurs d'arrondis ne donnent pas lieu à des résultats incohérents, et que les calculs puissent être effectués en un temps fini). La question de trouver de manière théorique le pas optimal peut s'avérer un problème épineux. Dans un premier temps, on peut se contenter de faire des tests pratiques comme dans l'exemple suivant.

Dans le script suivant, on applique la méthode d'Euler sur l'intervalle 01 avec la condition initiale $x(0) = 1$.

```python
from math import exp


print("{:^10} | {:^12} | {:^12} | {:^13}".format('h','x(t)','exp(t)','erreur'))
print('-'*57)
for i in range(1, 8):
    h, t, x  = 10**(-i), 0, 1
    while t < 1:
        t, x = t + h, x * (h + 1)
    print("{0:>10g} | {1:>.10f} | {2:>.10f} | {3:> .10f}"
            .format(h, x, exp(t),  exp(t) - x))
```

```
    h      |     x(t)     |    exp(t)    |    erreur
---------------------------------------------------------
       0.1 | 2.8531167061 | 3.0041660239 |  0.1510493178
      0.01 | 2.7048138294 | 2.7182818285 |  0.0134679990
     0.001 | 2.7169239322 | 2.7182818285 |  0.0013578962
    0.0001 | 2.7184177414 | 2.7185536702 |  0.0001359288
     1e-05 | 2.7182954199 | 2.7183090114 |  0.0000135915
     1e-06 | 2.7182804691 | 2.7182818285 |  0.0000013594
     1e-07 | 2.7182819660 | 2.7182820996 |  0.0000001336
     1e-08 | 2.7182817983 | 2.7182818347 |  0.0000000363
     1e-09 | 2.7182820738 | 2.7182818299 | -0.0000002438
```

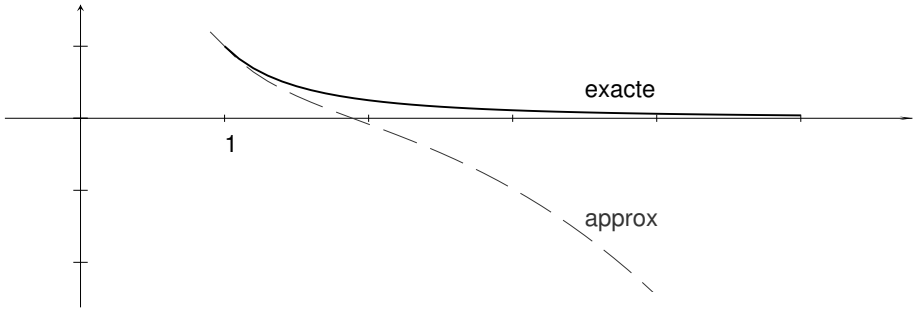
Pour un pas $h \lesssim 10^{-9}$, le temps de calcul devient trop important, de plus, on voit poindre l'effet des erreurs d'arrondis, puisque la solution approchée est devenu supérieure à l'exponentielle, ce qui est mathématiquement faux, en vertu de l'inégalité :

$$\forall n \in \mathbb{N}, \quad \left(1 + \frac{1}{n}\right)^n \leq e$$

En général, il ne suffit pas qu'un schéma numérique soit convergent pour qu'il donne de bons résultats sur n'importe quelle équation différentielle. Encore faut-il que le problème soit mathématiquement bien posé (en particulier, les hypothèses du théorème de CAUCHY-LIPSCHITZ doivent être vérifiées), qu'il soit numériquement bien posé (continuité suffisamment bonne par rapport aux conditions initiales) et qu'il soit bien conditionné (temps de calcul raisonnable).
Considérons par exemple le problème de CAUCHY :

$$\begin{cases} x(1) = 1 \\ x'(t) = 3\frac{x(t)}{t} - \frac{5}{t^3} \end{cases}$$

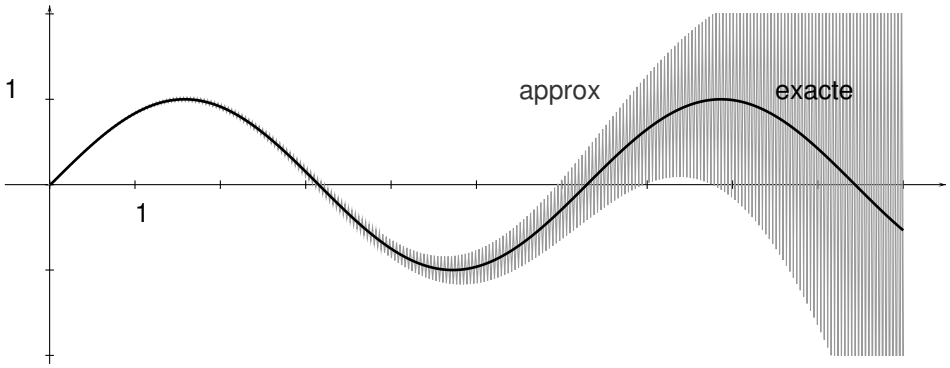
dont la solution est la fonction ? ? ?




On constate qu'ici la solution approchée s'écarte assez rapidement de la solution exacte $\varphi$. En effet, la forme générale des solutions de l'équation différentielle précédente est donnée par $t \mapsto \lambda x^3 + 1/x^2$. La condition $x(1) = 1$ impose $\lambda = 0$, mais dès qu'on s'écarte de la solution $\varphi$ on suit tangentiellement des courbes intégrales qui comportent un terme en $\lambda x^3$ et donc qui diffère notablement de la solution. Donc le problème est mal posé. Ici le schéma numérique n'est pas en cause.
Soit à résoudre à présent le problème de CAUCHY :

$$\begin{cases} x(1) = 1 \\ x'(t) = 100\,(\sin(t) - x) \end{cases}$$

dont la solution est la fonction $\varphi : t \mapsto \frac{1}{10001}(-100\cos t + 10000\sin t + 100\exp(-100\cdot t)) \approx \sin t$. Que se passe-t-il si on résout ce problème avec un pas de l'ordre de 0.02 ?




Avec un tel pas, la solution approchée oscille en s'éloignant de plus en plus de la solution exacte. En effet, le schéma numérique est donné par :

$$x_{k+1} = x_k + 100\,h\,(\sin t_k - x_k) = (1 - 100h)x_k + 100h\sin t_k$$

Donc une erreur de $\varepsilon_k$ sur $x_k$ aura les répercussions suivantes sur le calcul des termes ultérieurs :

$$\varepsilon_{k+1} = (1 - 100h)\varepsilon_k \implies \varepsilon_{k+n} = (1 - 100h)^n \varepsilon_k$$

Ainsi donc, tant que $1 - 100h \geq -1$, tant que $h \lesssim 0.002$, une petite erreur sur l'un des termes aura des répercussions allant en s'amplifiant.
Bien que le problème soit numériquement bien posé, il est en fait mal conditionné.

## Méthode de Runge-Kutta d'ordre 4

Si, comme nous l'avons déjà dit, la méthode d'Euler présente un intérêt théorique, on préfère en pratique des méthodes d'ordre plus élevé. Parmi la multitude des schémas numériques (méthodes à un pas comme celle de Taylor, méthodes à pas multiples comme celles d'Adams-Bashforth, d'Adams-Moulton, de prédiction-correction) celle qui présente le meilleur rapport précision/complexité est certainement celle de Runge-Kutta d'ordre 4.

En voici le schéma pour $\forall k \in [\![0, N]\!]$,

$$\begin{cases} t_{k+1} &= t_k + h \\ x_{k+1} &= x_k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{cases} \quad \text{où} \quad \begin{cases} k_1 &= h \cdot f(t_n, x_n) \\ k_2 &= h \cdot f\left(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}\right) \\ k_3 &= h \cdot f\left(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}\right) \\ k_4 &= h \cdot f(t_n + h, x_n + k_3) \end{cases}$$

Les coefficients qui apparaissent dans ces formules mystérieuses sont judicieusement ajustés pour obtenir une méthode d'ordre 4, sans pour autant avoir à calculer les dérivées successives de $f$ (comme c'est le cas dans les méthodes de Taylor), ou à recourir à une formule de récurrence d'ordre au moins 2 pour définir $x_k$ (comme c'est le cas dans les méthodes à pas multiples).

Définissez une fonction adaptée à ce schéma. Vous devriez observer pour l'exponentielle

$$\begin{cases} x'(t) = x(t) \\ x(0) = 1 \end{cases}$$

C

```
    h      |   erreur euler    |    erreur rk4
-----------------------------------------------------
      0.1  |  1.5104931784e-01 |  2.5338874514e-06
     0.01  |  1.3467999038e-02 |  2.2464341498e-10
    0.001  |  1.3578962232e-03 |  2.2204460493e-14
   0.0001  |  1.3592881559e-04 | -2.6645352591e-13
    1e-05  |  1.3591551171e-05 | -5.2180482157e-12
    1e-06  |  1.3591611072e-06 |  2.1464163780e-11
```

On constate que pour $h = 0.001$, la méthode d'Euler ne donne que trois décimales significatives du nombre $e$, alors que la méthode de Runge-Kutta en donne quatorze ! On remarque également qu'il est inutile d'appliquer la méthode de Runge-Kutta avec un pas $h \gtrsim 0.001$, puisque dans ce cas les erreurs d'arrondi prennent le dessus par rapport au gain théorique de précision.

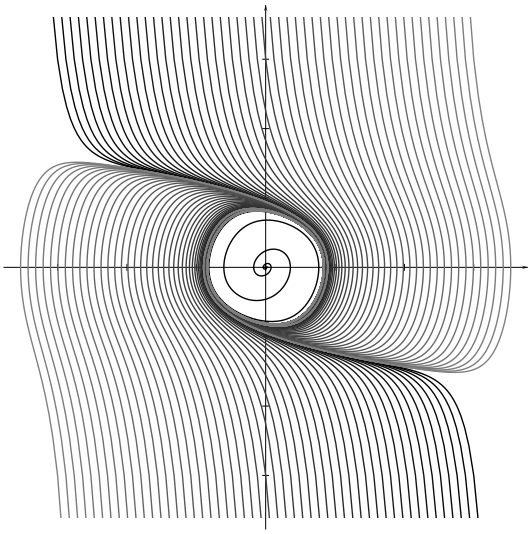
## Système

Soit à représenter les courbes intégrales du système autonome suivant, appelé oscillateur de van der Pol :

$$\begin{cases} \frac{\mathrm{d}x}{\mathrm{d}t} = y \\ \frac{\mathrm{d}y}{\mathrm{d}t} = \frac{y}{2} - x - y^3 \end{cases}$$

Les variables $x$, $k1$,... $k4$ représentent à présent des listes. Or une instruction comme $x + k1/2$ renverra un message d'erreur. En effet, la division d'une liste par un entier n'est pas définie ; de plus, l'addition de deux listes effectue la concaténation des listes, et non l'addition des listes composante par composante.

Il faut alors travailler non plus avec des listes, mais avec des objets supportant la vectorisation, comme c'est le cas des tableaux de la librairie de tierce partie **NumPy** ; en outre, cette option a l'avantage de réduire de manière significative les temps de calcul.



### Un exemple d'équation différentielle linéaire d'ordre 2

On rappelle qu'une équation différentielle linéaire d'ordre $n$ peut toujours s'écrire comme un système différentiel d'ordre 1. Illustrons ceci dans le cas $n = 2$ : en notant :

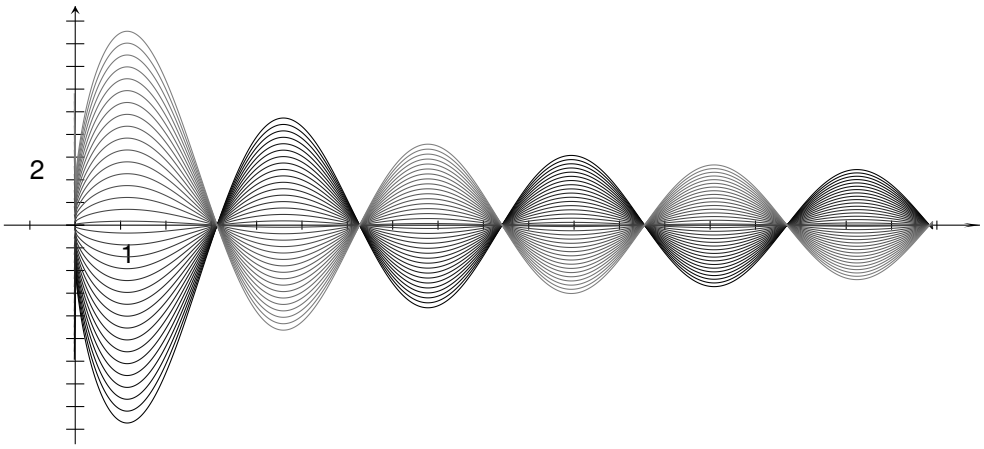
$$A(t) = \begin{pmatrix} 0 & 1 \\ -b(t) & -a(t) \end{pmatrix}, \quad X(t) = \begin{pmatrix} x(t) \\ x'(t) \end{pmatrix}, \quad C(t) = \begin{pmatrix} 0 \\ c(t) \end{pmatrix}$$

on a les équivalences

$$\begin{aligned} \forall t \in I \quad x'' + a(t)x' + b(t)x = c(t) &\iff \forall t \in I \quad \begin{cases} x' = x' \\ x'' = -bx - ax' + c \end{cases} \\ &\iff \forall t \in I \quad \frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x \\ x' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -b & -a \end{pmatrix} \cdot \begin{pmatrix} x \\ x' \end{pmatrix} + \begin{pmatrix} 0 \\ c \end{pmatrix} \\ &\iff \forall t \in I \quad X' = A(t) \cdot X + C(t) \end{aligned}$$

Considérons, par exemple, l'équation différentielle du 2e ordre dite de Bessel :

$$t^2x''(t) + tx'(t) + \left(t^2 - \alpha^2\right)x(t) = 0 \iff X'(t) = A(t) \cdot X(t) \quad \text{avec} \quad A(t) = \begin{pmatrix} 0 & 1 \\ \frac{\alpha^2 - t^2}{t^2} & -\frac{1}{t} \end{pmatrix}$$




## Lien avec les méthodes de quadrature

Version objet :

```
class Integration(object):
    def __init__(self, a, b, n):
        self.a, self.b, self.n = a, b, n
```

```
        self.poids_pivots = self.quadrature()

    def quadrature(self):
        raise NotImplementedError

    def integrale(self, f):
        return sum(w * f(x) for w, x in self.poids_pivots)

class Rectangles(Integration):
    def quadrature(self):
        a, b, n = self.a, self.b, self.n
        h = (b-a) / n
        return [(h, a + i*h) for i in range(n)]

class Trapezes(Integration):
    def quadrature(self):
        a, b, n = self.a, self.b, self.n
        h = (b-a) / n
        return [(h, a + (i+0.5)*h) for i in range(n)]

class Simpson(Integration):
    def coeff(self, i, N):
        if i == 0 or i == N:
            return 1
        elif i % 2 == 1:
            return 4
        else:
            return 2

    def quadrature(self):
        a, b, n = self.a, self.b, self.n
        if n % 2 != 1:
            n += 1
        h = (b-a) * 0.5 / n
        return [(h/3*self.coeff(i, 2*n), a + i*h) for i in range(2*n+1)]

class Romberg(object):
    def __init__(self, a, b, m):
        self.a, self.b, self.m = a, b, m

    def integrale(self, f):
        a, b, m = self.a, self.b, self.m
        A = [[(b-a) * (f(a)+f(b)) * 0.5]]
        for k in range(1, m+1):
            h = (b-a) / 2**k
            Ap = h * sum([f(a + j*h) for j in range(1, 2**k, 2)])
            A.append([0.5 * A[k-1][0] + Ap])
        for t in range(1, m+1):
            for k in range(t, m+1):
                A[k].append((4**t * A[k][t-1] - A[k-1][t-1]) / (4**t-1))
        return A[m][m]
```

Représenter la spirale de Cornu définie par

$$x(t) = \int_0^t \cos\left(u^2\right)\,\mathrm{d}u \quad \text{et} \quad y(t) = \int_0^t \sin\left(u^2\right)\,\mathrm{d}u$$

## SciPy

Les exercices suivants se proposent de reprendre quelques schémas numériques en utilisant cette fois-ci les primitives disponibles dans le module **SciPy**.

### Recherche F - 1 Intégration numérique

Utiliser la fonction **integrate** du module **scipy** pour tirer des formules suivantes une ap-

proximation de $\pi$ :

$$\pi = 4 \cdot I \quad \text{avec} \quad I = \int_0^1 \sqrt{1-t^2}\,\mathrm{d}t \qquad \text{et} \qquad \pi = 12\left(I - \frac{\sqrt{3}}{8}\right) \quad \text{avec} \quad J = \int_0^{\frac{1}{2}} \sqrt{1-t^2}\,\mathrm{d}t$$

Comparer les résultats avec la valeur de $\pi$ fournie par **`scipy`**.

## Recherche F - 2 Équations différentielles

En important la fonction **`integrate`** du module **`scipy`**, tracer les solutions des équations différentielles suivantes :

$$\underline{a)}\ x'(t) = 1 + t^2 \cdot x(t)^2 \qquad \underline{b)}\ \frac{\mathrm{d}x}{\mathrm{d}t} = y\frac{\mathrm{d}y}{\mathrm{d}t} = \frac{y}{2} - x - y^3 \qquad \underline{c)}\ t^2x''(t) + tx'(t) + \left(t^2 - \alpha^2\right)x(t) = 0$$