[MN] Práctica 2 - Apunte

September 7, 2021

Métodos numéricos

Segundo cuatrimestre 2021

Práctica 2: Esquemas temporales

Cátedra: Pablo Dmitruk

1 Esquemas temporales

1.1 Integración temporal

En esta práctica nos interesa poder resolver numéricamente el problema

$$\begin{aligned} \dot{y} &= f(t, y), \\ f(t_0) &= y_0, \end{aligned}$$

donde para alivianar la notación usaremos en adelante $\dot{y} = \mathrm{d}y/\mathrm{d}t$.

Noten que, a diferencia de lo que vimos en la práctica anterior, conocemos ahora la derivada y queremos recuperar y^n (es decir, el valor de y sobre una cantidad discreta de valores de t, t^n). Luego, asumiendo conocidos $y^j, f(t^j, y^j)$ para $0 \leq j \leq n$, una forma de determinar y^{n+1} es usar expansiones de Taylor. Por ejemplo, expandiendo y^{n+1} alrededor de y^n o bien y^n alrededor de y^{n+1}, tenemos respectivamente

$$\begin{aligned} y^{n+1} &= y^n \quad + \dot{y}^n k \quad + \mathcal{O}(k^2), \\ y^n &= y^{n+1} - \dot{y}^{n+1} k + \mathcal{O}(k^2), \end{aligned}$$

donde $k = \Delta t$ es el paso temporal y $f^n = f(t^n, y^n)$. Podemos despejar fácilmente sendos métodos de segundo **orden local**

$$y^{n+1} = y^n + f^n k, \qquad \text{(Euler adelantado)}$$

$$y^{n+1} = y^n + f^{n+1} k. \qquad \text{(Euler atrasado)}$$

Como indican las etiquetas, estos métodos reciben el nombre de **Euler adelantado** y **Euler atrasado**, respectivamente. Estos métodos determinan y^{n+1} a segundo orden, si conocemos y^n y f^n o f^{n+1} de manera exacta (o, al menos, a orden k^2 y k, respectivamente). Sin embargo, este no suele ser el caso, ya que lo usual suele ser partir de una condición inicial y aplicar integradores temporales de forma iterativa hasta llegar a la solución a un cierto tiempo final t_f deseado. Al aplicar el método $N = t_f/k$ veces, el orden será entonces $N\mathcal{O}(k^2) = t_f\mathcal{O}(k)$. En consecuencia, el **orden**

global, es decir, el error asociado a integrar hasta un cierto tiempo t, es $\mathcal{O}(k)$ para los métodos de Euler (adelantado y atrasado). Encontraremos esta diferencia de un orden de magnitud entre el orden local y el orden global para todos los esquemas temporales que veremos, al menos mientras permanezcan **estables**, concepto que discutiremos más adelante. Excepto mención explícita de lo contrario, **cuando nos referimos al orden de un integrador temporal hacemos referencia al orden global** del mismo.

Un elemento en el que aún no reparamos es en que si conocemos funcionalmente f, y contamos con y^n, la ecuación de Euler hacia adelante resulta inmediata para resolver. Sin embargo, la versión atrasada, es una expresión implícita para y^{n+1} (ya que f^{n+1} depende de y^{n+1}). Esto último no presenta gran dificultad (en precisión infinita) si f^{n+1} es lineal, pero no resulta trivial en casos generales, requiriendo generalmente usar métodos iterativos para hallar y^{n+1}. A los métodos que, conocida toda la información para t^j, y^j, $j \leq n$ permiten determinar explícitamente y^{n+1} se los llama **métodos explícitos**. Por otro lado, aquellos que requiren resolver una ecuación implicita (como Euler hacia atrás), se los llama **métodos implícitos**. A la hora de utilizar métodos implícitos, es usual conjugarlos con algún esquema iterativo, como por ejemplo el algoritmo de Newton-Raphson hasta alcanzar la tolerancia deseada. Así planteado, parecería que los métodos implícitos son una complicación innecesaria, sin embargo a la hora de discutir estabilidad, veremos que pueden ser muy ventajosos para algunos problemas, mientras que los explícitos serán una opción mejor en otros casos.

Para construir métodos de mayor orden, las estrategias más populares (y que veremos en el curso) incluyen: - **Métodos multipaso lineales**: en forma similar a lo que hacíamos con diferencias finitas, se propone y^{n+1} como función de múltiples valores y^j y f^j, en lugar de usar solo y^n y f^n/f^{n+1} como hicimos para los métodos de Euler. - **Métodos predictores-correctores**: generan una solución aproximada a partir de métodos de bajo orden (la *predicción*) que luego es mejorada usando algún algoritmo de interpolación (la *corrección*). - **Métodos de Runge-Kutta**: utilizan, en lugar de valores de y y de f para pasos previos/posteriores, valores intermedios de f, como por ejemplo $f^{n+\frac{1}{2}} = f\left(t^n + \frac{k}{2}, y\left(t^n + \frac{k}{2}\right)\right)$. Pueden verse como un caso de métodos predictores-correctores, aunque su uso extendido hace que reciban una denominación propia.

Veamos, antes de pasar a considerar en detalle estas estrategias, por qué nos concentraremos solo en integrar y a partir de su derivada primera $\dot{y}$. Spoiler: una EDO de orden N puede reducirse a un sistema de N EDOs de primer orden. Vale notar, sin embargo, que existen integradores temporales para EDOs de orden superior, aunque no los veremos en el curso. Cuando en prácticas posteriores veamos ecuaciones en derivadas parciales EDPs, trabajar con integradores para la primer derivada temporal tampoco resultará un limitante, como oportunamente mostraremos.

1.1.1 Reducción de dimensionalidad de una EDO (ecuación diferencial ordinaria)

Si bien en la materia solo veremos integradores temporales para sistemas de EDOs de primer orden, esto no representa restricción alguna con respecto a los problemas que podremos resolver. Como probablemente hayan visto en materias anteriores, cualquier EDO de orden N puede reescribirse como un sistema de N EDOs de primer orden. La manera más simple de verlo es con un ejemplo. Dada el problema de valores iniciales de orden 2

$$\dot{y}\ddot{y}\cos(t) + t^2\dot{y}^2 = y\ln(y), \qquad y(t_0) = y_0 \quad \wedge \quad \dot{y}(t_0) = \dot{y}_0,$$

podemos substituir $u_1 = y$, $u_2 = \dot{y}$, y de esta manera tenemos el sistema

$$\begin{aligned} \dot{u}_1 &= u_2, & u_1(t_0) &= y_0, \\ \dot{u}_2 &= \frac{u_1 \ln(u_1) - t^2 u_2^2}{u_2 \cos(t)}, & u_2(t_0) &= \dot{y}_0, \end{aligned}$$

que es un sistema de 2 ecuaciones diferenciales de primer orden acopladas expresable como

$$\begin{aligned} \dot{\mathbf{u}} &= \mathbf{F}(t, \mathbf{u}), \\ \mathbf{u}(t_0) &= \mathbf{u_0}, \end{aligned}$$

con

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}, \qquad \mathbf{u}_0 = \begin{pmatrix} y_0 \\ \dot{y}_0 \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} u_2 \\ \dfrac{u_1 \ln(u_1) - t^2 u_2^2}{u_2 \cos(t)} \end{pmatrix}.$$

Vemos entonces que, reescribiendo nuestro problema, los métodos que veremos a continuación resultan fácilmente aplicables a ecuaciones de órdenes superiores. Además, para alivianar la notación, presentaremos los esquemas de integración temporal considerando una única EDO de primer orden. Sin embargo, todos los métodos son fácilmente generalizables a sistemas de EDOs, como veremos al final con un ejemplo.

1.1.2 Métodos multipaso lineales

Este tipo de métodos se llaman lineales ya que proponen una combinación lineal (de allí el nombre lineal) de y^j y f^j usando información correspondiente a varios pasos temporales (de allí el nombre multipaso). Esto quiere decir que, en general, podemos escribirlos para un cierto tiempo t^n como

$$\sum_{j=0}^{s} \alpha^j y^{n+j} = h \sum_{j=0}^{s} \beta^j f^{n+j}, \tag{1}$$

dando valores específicos de $\boldsymbol{\alpha}$ y $\boldsymbol{\beta}$ a realizaciones particulares de métodos multipaso lineales. Si bien a simple vista la elección de $\boldsymbol{\alpha}$ y $\boldsymbol{\beta}$ puede parecer trivial, i.e. ajustar los $2s + 2$ grados de libertad de forma de obtener esquemas de orden $\mathcal{O}(k^{2s})$, si buscamos obtener integradores estables (veremos por qué son los únicos que nos interesan más adelante), el mayor orden de aproximación que podemos obtener con la ecuación (1) es $\mathcal{O}(k^{s+2})$ (Teorema de Dahlquist). De esta manera, aparecen muchas propuestas de cómo ajustar los correspondientes s grados de libertad.

Una dificultad que presentan los métodos multipaso es que, por construcción, **resulta necesario conocer y^j y f^j para múltiples instantes**, información con la que no contamos si solo disponemos de una condición inicial y^0. **En la práctica, una manera usual de resolver esto es inicializar la integración con otro método** (ej: Euler con alta resolución temporal —de forma de mantener el orden de aproximación—, Runge-Kutta, predictor-corrector, etc.). Una vez contamos con la cantidad de pasos suficientes comenzamos la aplicación del método multipaso deseado.

En la práctica de la materia veremos dos familias de métodos multipaso lineales: los métodos de **Adams-Bashforth** (explícitos) y los de **Adams-Moulton** (implícitos). Ambos métodos en su conjunto se conocen como métodos de *Adams*[†] y están dados por la expresión

$$y^{n+s} - y^{n+s-1} = k \sum_{j=0}^{s} \beta^j f^{n+j},$$

es decir, surgen de escoger $\alpha_0, \ldots, \alpha_{s-2} = 0$, $\alpha_{s-1} = -1$ y $\alpha_s = 1$.

†: el mismo Adams que postuló por primera vez la existencia de Neptuno.

Adams-Bashforth En el método de Adams-Bashforth se busca obtener un método que resulte explícito y por lo tanto se propone $\beta_s = 0$, es decir

$$y^{n+s} - y^{n+s-1} = k \sum_{j=0}^{s-1} \beta^j f^{n+j} \qquad \text{(Adams-Bashforth)}$$

De esta forma solo queda determinar los coeficientes $\beta^n, \ldots, \beta^{n+s-1}$ y con ello tendremos ya una fórmula que permite determinar y^{n+s} si conocemos y^{n+s-1} y $f^n, \ldots, f^{n+s-1}$. Para ello, naturalmente, buscaremos un método consistente y maximizar el orden de aproximación a y^{n+s}. Para esto consideremos Y una solución exacta a la EDO en cuestión, reemplazando en la última ecuación tenemos

$$\left[Y + \dot{Y}k + \ddot{Y}\frac{k^2}{2} + \ldots\right]_{t^{n+s-1}} - Y^{n+s-1} + \delta Y = k\Bigg\{\beta^{s-1} f^{n+s-1} + \beta^{s-2}\left[f - \dot{f}k + \ddot{f}\frac{k^2}{2} + \ldots\right]_{t^{n+s-1}} + \\ + \ldots + \beta^0\left[f - \dot{f}(s-1)k + \ddot{f}\frac{(s-1)^2k^2}{2} + \ldots\right]_{t^{n+s-1}}\Bigg\}$$

donde reemplazamos Y^{n+s} por su expansión de Taylor alrededor de Y^{n+s-1}, f^{n+s-j} por las correspondientes expansiones alrededor de f^{n+s-1}, y δY es el error de truncamiento. Regrupando obtenemos

$$\delta Y = k\dot{Y}^{n+s-1}\left[\beta^{s-1} + \beta^{s-2} + \ldots + \beta^0 - 1\right] + \; + k^2\ddot{Y}\left[-\beta^{s-2} - \ldots - (s-1)\beta^0 - \frac{1}{2}\right] + \ldots, \qquad \text{(Error de truncamiento)}$$

donde usamos $f^j = \dot{Y}^j$, $\dot{f}^j = \ddot{Y}^j$ y así sucesivamente.

Vemos que para que haya consistencia, debemos pedir $\sum_j \beta^j = 1$, este resultado es general y aplica a todos los métodos multipaso lineales. Los restantes s coeficientes se obtienen intentando cancelar la mayor cantidad de potencias de k, de forma de conseguir un método con el mejor orden de aproximación posible para la cantidad de pasos en consideración.

Deducción de coeficientes para un método de 3 pasos Para ver un ejemplo de esto, intentemos deducir el método de Adams-Bashforth que utiliza 3 pasos ($s = 3$). Es decir, debemos determinar $\beta^0, \beta^1, \beta^2$ ($\beta^3 = 0$ por ser explícito, i.e. un método de Adams-Bashforth). Para ello, escribimos el caso partícular de la ecuación anterior

$$\delta Y = k\dot{Y}\left[\beta^2 + \beta^1 + \beta^0 - 1\right] + k^2\ddot{Y}\left[-\beta^1 - 2\beta^0 - \frac{1}{2}\right] + \\ + k^3\dddot{Y}\left[\frac{\beta^1}{2} + \frac{2^2}{2}\beta^0 - \frac{1}{6}\right] + \mathcal{O}(k^4),$$

y pidiendo que se anulen la mayor cantidad de órdenes dominantes posibles (i.e., lo que acompaña a k, k^2, k^3) tenemos

$$\begin{pmatrix} 1 & 1 & 1 \\ -2 & -1 & 0 \\ 2 & \frac{1}{2} & 0 \end{pmatrix}\begin{pmatrix} \beta^0 \\ \beta^1 \\ \beta^2 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{6} \end{pmatrix},$$

que tiene como solución

$$\beta^0 = \frac{5}{12}, \qquad \beta^1 = -\frac{16}{12}, \qquad \beta^2 = \frac{23}{12},$$

obteniendo un método donde $\delta F = \mathcal{O}(k^4)$, o sea, un método de orden 3 (recuerden que el orden global es un orden de magnitud menor que el error al cabo de un paso). En general, un método de Adams-Bashforth de s pasos, posee un orden (global) s.

En una notación más operativa, podemos escribir el resultado hallado como

$$y^{n+1} = y^n + \frac{k}{12}\left(23f^n - 16f^{n-1} + 5f^{n-2}\right).$$

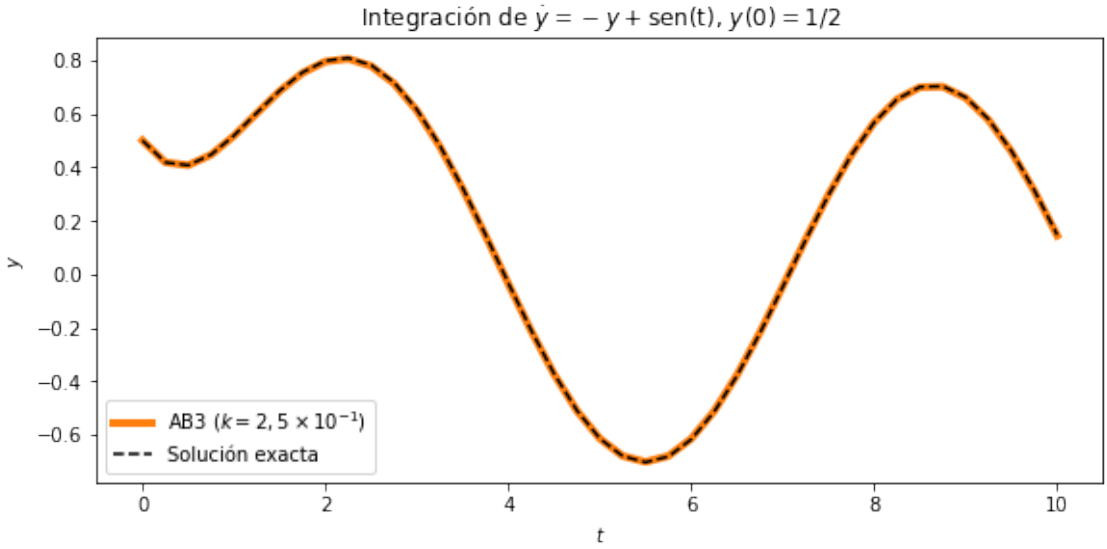
Veamos en acción el método que acabamos de derivar. Para eso consideramos el problema de valor inicial

$$\dot{y}(t) = -y(t) + \text{sen}(t), \qquad y(0) = 1/2,$$

para $0 \leq t \leq 10$, que tiene como solución analítica $y(t) = e^{-t} + [\text{sen}(t) - \cos(t)]/2$. Nos valdremos de esta solución analítica para resolver el problema de la inicialización. Esta estrategia no será posible cuando desconozcamos la solución analítica, debiendo recurrir a inicializar la integración con otros métodos, como mencionamos anteriormente.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

dt    = 2.5e-1              # Paso temporal
y0    = 1/2                 # Condición inicial
tf    = 10                  # Tiempo final de integración
pasos = int(round(tf/dt))   # Cantidad de pasos

y     = np.zeros( pasos+1 )  # Variable para ir guardando la integración

# Agrego los tres primeros pasos
y[0] = y0
y[1] = np.exp(-dt)   + (np.sin(dt)  - np.cos(dt) )/2
y[2] = np.exp(-2*dt) + (np.sin(2*dt)-np.cos(2*dt))/2

# Integro usando AB3
for n in range(2, pasos):
    tn  =     n*dt  # t^n
    tn1 = (n-1)*dt  # t^{n-1}
    tn2 = (n-2)*dt  # t^{n-2}

    fn  = -y[n]   + np.sin(tn)   # f^n
    fn1 = -y[n-1] + np.sin(tn1)  # f^{n-1}
    fn2 = -y[n-2] + np.sin(tn2)  # f^{n-2}

    y[n+1] = y[n] + (23*fn - 16*fn1 + 5*fn2)*dt/12  # Integración explícita

# Grafico
t = np.arange(0, y.size)*dt
```

```
fig, ax = plt.subplots(1, 1, figsize=(8,4), constrained_layout=True)
ax.plot(t, y, label=r"AB3 ($k=2,5 \times 10^{-1}$)", c="C1", lw=4)
ax.plot(t, np.exp(-t) + (np.sin(t)-np.cos(t))/2, "--k", label="Solución exacta")
ax.legend()
ax.set_title(r"Integración de $\dot{y} = -y + \mathrm{sen(t)}$, $y(0)=1/2$")
ax.set_xlabel("$t$")
ax.set_ylabel("$y$");
```



Resumen de métodos de Adams-Bashforth Incluimos a continuación una tabla con los coeficientes que acompañan a cada evaluación de f para los métodos de Adams-Bashforth hasta orden 4.

Pasos	Orden	f^n	f^{n-1}	f^{n-2}	f^{n-3}
1	1	1	0	0	0
2	2	$\frac{3}{2}$	$-\frac{1}{2}$	0	0
3	3	$\frac{23}{12}$	$-\frac{16}{12}$	$\frac{5}{12}$	0
4	4	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$

Noten que para $s = 1$ recuperamos el método de Euler hacia adelante.

Adams-Moulton En los métodos de Adams-Moulton se sigue la misma lógica que la empleada en Adams-Bashforth, pero se relaja la restricción $\beta^s = 0$, resultando por tanto en un método implícito. La forma de obtener los coeficientes apropiados es completamente análoga, obteniendo la siguiente relación para el error de truncamiento

$$\delta Y = k\dot{Y}^{n+s-1}\left[\beta^s + \beta^{s-1} + \beta^{s-2} + \ldots + \beta^0 - 1\right] + k^2\ddot{Y}\left[\beta^s - \beta^{s-2} - \ldots - (s-1)\beta^0 - \frac{1}{2}\right] + \ldots,$$

que naturalmente tiene un grado de libertad mayor a la expresión hallada para Adams-Bashforth, dado por β^s.

Deducción de coeficientes para un método de 3 pasos De manera análoga a lo que hicimos para el caso explícito, calculemos los coeficientes para el método implícito (Adams-Moulton) de 3 pasos. Esto implica hallar β^3, β^2, β^1 y β^0. Para ello escribamos el error de truncamiento hasta un orden apropiado

$$\delta Y = k\dot{Y}\left[\beta^3+\beta^2+\beta^1+\beta^0-1\right]+k^2\ddot{Y}\left[\beta^3-\beta^1-2\beta^0-\frac{1}{2}\right]+$$
$$+k^3\dddot{Y}\left[\frac{\beta^3}{2}+\frac{\beta^1}{2}+\frac{2^2}{2}\beta^0-\frac{1}{6}\right]++k^4\ddot{Y}\left[\frac{\beta^3}{6}-\frac{\beta^1}{6}-\frac{2^3}{6}\beta^0-\frac{1}{24}\right]+\mathcal{O}(k^5).$$

Pedimos nuevamente que se anulen la mayor cantidad de órdenes dominantes posibles, es decir, lo que acompaña a k, k^2, k^3 y k^4. Noten que como tenemos un coeficiente más, podemos anular un término más con respecto a lo planteado para Adams-Bashforth. Tenemos entonces

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 \\ 2 & \frac{1}{2} & 0 & \frac{1}{2} \\ -\frac{4}{3} & -\frac{1}{6} & 0 & \frac{1}{6} \end{pmatrix}\begin{pmatrix}\beta^0\\ \beta^1\\ \beta^2\\ \beta^3\end{pmatrix}=\begin{pmatrix}1\\ \frac{1}{2}\\ \frac{1}{6}\\ \frac{1}{24}\end{pmatrix},$$

que tiene como solución

$$\beta^0=\frac{1}{24},\qquad \beta^1=-\frac{5}{24},\qquad \beta^2=\frac{19}{24},\qquad \beta^3=\frac{9}{24}$$

obteniendo un método 5. En general, un método de Adams-Moulton de s pasos, posee un orden (global) $s+1$.

En una notación más operativa, podemos escribir el resultado hallado como

$$y^{n+1}=y^n+\frac{k}{24}\left(9f^{n+1}+19f^n-5f^{n-1}+f^{n-2}\right),$$

que es una expresión implícita para y^{n+1}.

Al igual que hicimos antes, consideremos el problema de valores iniciales

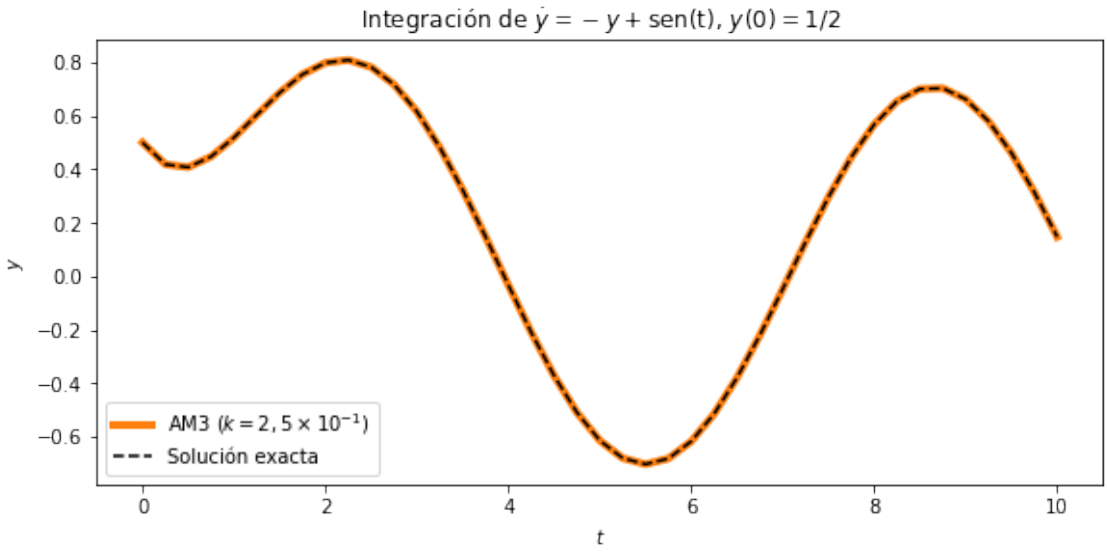
$$\dot{y}(t)=-y(t)+\operatorname{sen}(t),\qquad\qquad y(0)=1/2,$$

para $0\leq t\leq 10$. Al igual que antes, haremos uso de la solución analítica $y(t)=e^{-t}+[\operatorname{sen}(t)-\cos(t)]/2$ para inicializar la integración. Además, dado que esta ecuación es lineal, el hecho de que el método sea implícito no supondrá dificultad, ya que puede realizarse el despeje de la siguiente manera

$$\begin{aligned} y^{n+1} &= y^n+\frac{k}{24}\left[9(-y^{n+1}+\operatorname{sen}(t^{n+1})+19f^n-5f^{n-1}+f^{n-2}\right] \\ &= \left[y^n+\frac{k}{24}\left(9\operatorname{sen}(t^{n+1})+19f^n-5f^{n-1}+f^{n-2}\right)\right]\frac{1}{1+\frac{9}{24}k}. \end{aligned}$$

Esto no generalizará a problemas no lineales, donde habrá que usar algún esquema iterativo para hallar y^{n+1}.

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

dt    = 2.5e-1              # Paso temporal
y0    = 1/2                 # Condición inicial
tf    = 10                  # Tiempo final de integración
pasos = int(round(tf/dt))   # Cantidad de pasos

y     = np.zeros( pasos+1 )  # Variable para ir guardando la integración

# Condición inicial y primeras iteraciones (las saco de la solución analítica)
y[0] = y0
y[1] = np.exp(-dt)   + (np.sin(dt)  - np.cos(dt) )/2
y[2] = np.exp(-2*dt) + (np.sin(2*dt)-np.cos(2*dt))/2

# Integro usando AM4
for n in range(2, pasos):
    ts  = (n+1)*dt  # t^{n+1}  (tiempo siguiente)
    tn  =     n*dt  # t^n
    tn1 = (n-1)*dt  # t^{n-1}
    tn2 = (n-2)*dt  # t^{n-2}

    fn  = -y[n]   + np.sin(tn)   # f^n
    fn1 = -y[n-1] + np.sin(tn1)  # f^{n-1}
    fn2 = -y[n-2] + np.sin(tn2)  # f^{n-2}

    y[n+1] = (y[n] + (9*np.sin(ts) + 19*fn - 5*fn1 + fn2)*dt/24)/(1+9/24*dt)

# Grafico
t = np.arange(0, y.size)*dt
fig, ax = plt.subplots(1, 1, figsize=(8,4), constrained_layout=True)
ax.plot(t, y, label=r"AM3 ($k=2,5 \times 10^{-1})$", c="C1", lw=4)
ax.plot(t, np.exp(-t) + (np.sin(t)-np.cos(t))/2, "--k", label="Solución exacta")
ax.legend()
ax.set_title(r"Integración de $\dot{y} = -y + \mathrm{sen(t)}$, $y(0)=1/2$")
ax.set_xlabel("$t$")
ax.set_ylabel("$y$");
```



Pasos	Orden	f^{n+1}	f^n	f^{n-1}	f^{n-2}	f^{n-3}
1	1	1	0	0	0	0
1	2	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0
2	3	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$	0	0
3	4	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$	0
4	5	$\frac{251}{720}$	$\frac{646}{720}$	$-\frac{264}{720}$	$\frac{106}{720}$	$-\frac{19}{720}$

Resumen de métodos de Adams-Moulton Vemos que el caso de 1 paso resulta degenerado, teniendo dos esquemas posibles. El de orden 1 no es ni más ni ménos que el método de Euler atrasado, mientras que el esquema de segundo orden se conoce como regla trapezoidal.

Otros métodos multipaso lineales Sin demostraciones de por medio, podemos mencionar otras familias populares de métodos multipaso lineales. Una opción por ejemplo es buscar operadores temporales centrados, que resulten de la forma

$$y^{n+s} - y^{n+s-2} = \sum_{j=0}^{s} \beta^j f^{n+j}.$$

Los métodos explícitos ($\beta^s = 0$) asociados a esta elección reciben el nombre de *métodos de Nyström*, mientras que a aquellos implícitos se los conoce como métodos de *Milne-Simpson*, y tienen orden s y $s+1$, respectivamente. Un método de Nyström que vieron en las clases teóricas es aquel conocido como ***Salto de rana*** (o *leapfrog* en inglés), dado por

$$y^{n+1} = y^{n-1} + 2kf^n,$$

que como vieron en las clases teóricas presenta un modo computacional (es decir, un modo *no-físico*).

Adicionalmente, y de manera opuesta a lo que proponen los métodos de Adams, pueden buscarse métodos con $\beta^0 = \beta^1 = \ldots = \beta^{s-1} = 0$, donde se ajusta β^s y $\boldsymbol{\alpha}$. Estos esquemas reciben el nombre de ***fórmulas de diferenciación hacia atrás*** (o *backward differentiation formulas* — BDF en inglés—).

1.1.3 Métodos predictores-correctores

Como mencionamos anteriormente, los métodos predictores-correctores consisten en usar dos integradores temporales y combinarlos. La técnica más común es la que combina un método explícito con uno implícito, de la siguiente manera

1. **Paso predictor (P)**: Usamos un método explícito para obtener una estimación de y^{n+1}, que llamamos y^{*n+1} por ejemplo, utilizando Euler adelantado tendremos:

$$y^{*n+1} = y^n + kf^n.$$

2. **Paso evaluador (E)**: A partir de la estimación para y^{n+1} obtenida en el paso anterior, calculamos $f(t^{n+1}, y^{*n+1})$, es decir, f^{*n+1}.

3. **Paso corrector (C)**: Llamando f^{*n+1} a $f(t^{n+1}, y^{*n+1})$, resolvemos para un esquema implícito. Utilizando ahora como ejemplo Euler atrasado, se obtiene:

$$y^{n+1} = y^n + kf^{*n+1} = y^n + kf(t^{n+1}, y^n + kf^n). \qquad \text{(Matsuno)}$$

   El método que acabamos de hallar es conocido como ***método de Matsuno*** y es $\mathcal{O}(k)$. Análogamente podríamos haber utilizado el esquema trapezoidal (Adams-Moulton de 2do orden) en lugar de Euler atrasado y tendríamos

$$y^{n+1} = y^n + \frac{k}{2}\left(f^{*n+1} + f^n\right) = y^n + \frac{k}{2}\left[f(t^{n+1}, y^n + kf^n) + f(t^n, y^n)\right], \qquad \text{(Heun)}$$

   conocido como ***método de Heun***, cuyo orden global es $\mathcal{O}(k^2)$.

Noten que en ambos casos acabamos con métodos explícitos, derivados de la combinación de uno explícito y uno implícito.

Vale mencionar que es posible realizar este proceso de manera iterativa, utilizando las ecuaciones de (Matsuno) y (Heun) como nuevos candidatos y^{*n+1}, obteniendo un nuevo valor para f^{*n+1} y volviendo a aplicar el método respectivo para obtener y^{n+1}. Si realizamos la operación de corrección c veces, suele notarse al método resultante como $P(EC)^c$.

Si bien los métodos predictores-correctores pueden estudiarse más formalmente, no lo haremos en el curso. Solo mencionaremos que otra combinación común para generar métodos predictores-correctores es la conjunción de métodos de Adams-Bashforth (predictor) y Adams-Moulton (corrector), que reciben el nombre de *métodos Adams-Bashforth-Moulton*.

1.1.4 Métodos de Runge-Kutta

A diferencia de los métodos multipaso, que utilizan valores de y y f calculados en pasos anteriores (o para el paso posterior si son implícitos), la idea de los métodos de Runge-Kutta es utilizar múltiples evaluaciones de f entre t^n y t^{n+1} (denominadas *etapas*) para generar una aproximación de mayor orden.

A priori, esto podría parecer ineficiente ya que en los métodos multipaso usamos información ya conocida, mientras que en Runge-Kutta estaremos utilizando nuevas evaluaciones que emplearemos solo para el paso en cuestión. Sin embargo, los métodos de Runge-Kutta precisan menos almacenamiento en memoria. Además presentan distintas características de estabilidad con respecto a los métodos multipaso, por lo que pueden ser más apropiados para algunas EDOs.

Los métodos de Runge-Kutta se clasifican de acuerdo a la cantidad de etapas (así como los multipaso a la cantidad de pasos utilizados). Por ejemplo, para un método de dos etapas tendremos

$$y^{n+1} = y^n + k\left(\alpha_1 f_1^n + \alpha_2 f_2^n\right),$$

donde $f_1^n = f(t_1^n, y_1^n)$ y $f_2^n = f(t_2^n, y_2^n)$ son evaluaciones de f entre t^n y t^{n+1}, es decir, $t^n \leq t_1^n \leq t_2^n \leq t^{n+1}$. La misma idea aplica para métodos con más etapas. ***Vale resaltar que veremos solo los métodos de Runge-Kutta explícitos ya que son los más empleados***.

En general, y a diferencia de los métodos multipaso, en el caso de Runge-Kutta no alcanza con definir una cantidad de etapas s y buscar maximizar el orden de aproximación para obtener un método único. Sin embargo, si nos restringimos a métodos cuyas evaluaciones de f están equiespaciadas (aunque en algunos casos repetidas), obtenemos los siguientes esquemas:

- ***Segundo orden***:

$$\begin{aligned} y^{n+1} &= y^n + R_2, \\ R_1 &= kf\left(t^n, y^n\right), \\ R_2 &= kf\left(t^n + \frac{k}{2}, y^n + \frac{R_1}{2}\right) \end{aligned} \tag{Midpoint}$$

$$\begin{aligned} y^{n+1} &= y^n + \frac{1}{2}\left(R_1 + R_2\right), \\ R_1 &= kf\left(t^n, y^n\right), \\ R_2 &= kf\left(t^n + k, y^n + R_1\right) \end{aligned} \tag{Heun}$$

- ***Tercer orden***:

$$\begin{aligned} y^{n+1} &= y^n + \frac{1}{4}\left(R_1 + 3R_3\right), \\ R_1 &= kf\left(t^n, y^n\right), \\ R_2 &= kf\left(t^n + \frac{k}{3}, y^n + \frac{R_1}{3}\right), \\ R_3 &= kf\left(t^n + 2\frac{k}{3}, y^n + 2\frac{R_2}{3}\right). \end{aligned} \tag{RK3}$$

- ***Cuarto orden***:

$$
\begin{aligned}
y^{n+1} &= y^n + \frac{1}{6}\left(R_1 + 2R_2 + 2R_3 + R_4\right), \\
R_1 &= kf\left(t^n, y^n\right), \\
R_2 &= kf\left(t^n + \frac{k}{2}, y^n + \frac{R_1}{2}\right), \\
R_3 &= kf\left(t^n + \frac{k}{2}, y^n + \frac{R_2}{2}\right), \\
R_4 &= kf\left(t^n + k, y^n + R_3\right).
\end{aligned}
\tag{RK4}
$$

Vemos que tenemos dos métodos de segundo orden. El primero de ellos llamado ***Euler mejorado*** o ***Punto medio*** (*midpoint* en inglés), mientras que el otro es el ya familiar método de Heun. Esto último muestra, como mencionamos anteriormente, que aunque no es del todo usual, es posible considerar a los métodos de Runge-Kutta como esquemas predictores-correctores, solo que operando sobre instantes intermedios (i.e. etapas) en lugar de entre pasos.

Adicionalmente, vemos que nuevamente recuperamos el método de Euler adelantado como el caso degenerado de un método Runge-Kutta de 1 etapa.

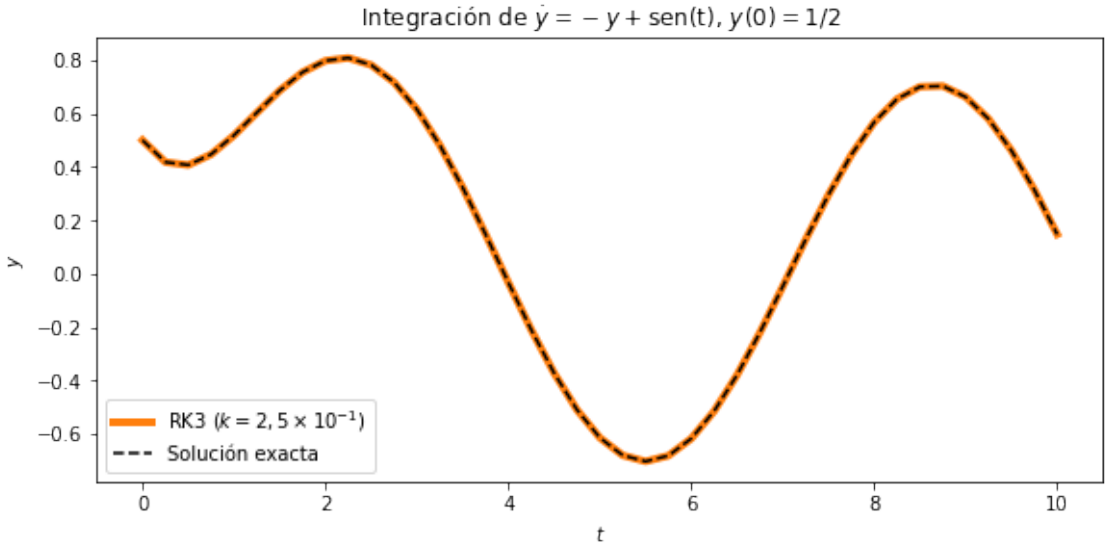
Al igual que hicimos antes, para ilustrar el uso de los métodos de Runge-Kutta, integremos el problema de valores iniciales

$$\dot{y}(t) = -y(t) + \operatorname{sen}(t), \qquad y(0) = 1/2,$$

para $0 \leq t \leq 10$ usando un método de Runge-Kutta de orden 3.

[ ]:
```
import numpy as np

dt    = 2.5e-1              # Paso temporal
y0    = 1/2                 # Condición inicial
tf    = 10                  # Tiempo final de integración
pasos = int(round(tf/dt))   # Cantidad de pasos

# Variable para ir guardando la integración
y    = np.zeros( pasos+1 )
y[0] = y0

# Integro usando RK3
for n in range(0, pasos):
    t1 = n*dt
    R1 = dt*(-y[n] + np.sin(t1))              # Primer etapa

    t2 = t1 + dt/3
    R2 = dt*(-(y[n] + R1/3) + np.sin(t2))     # Segunda etapa

    t3 = t2 + dt/3
    R3 = dt*(-(y[n] + 2*R2/3) + np.sin(t3))   # Última etapa
```

```
    y[n+1] = y[n] + (R1 + 3*R3)/4              # Combino las etapas

# Grafico
t = np.arange(0, y.size)*dt
fig, ax = plt.subplots(1, 1, figsize=(8,4), constrained_layout=True)
ax.plot(t, y, label=r"RK3 ($k=2,5 \times 10^{-1}$)", c="C1", lw=4)
ax.plot(t, np.exp(-t) + (np.sin(t)-np.cos(t))/2, "--k", label="Solución exacta")
ax.legend()
ax.set_title(r"Integración de $\dot{y} = -y + \mathrm{sen(t)}$, $y(0)=1/2$")
ax.set_xlabel("$t$")
ax.set_ylabel("$y$");
```



1.1.5 Aplicación a una EDO de orden superior

Consideremos ahora el siguiente problema de valores iniciales

$$\ddot{y} + \frac{2}{t^2+1}(y - t\dot{y}) = \left(\cos(t) + t\mathrm{sen}(t)\right)\frac{2}{t^2+1} - \cos(t), \qquad y(0) = 2, \qquad \dot{y}(0) = 0.$$

Este problema tiene como solución $y(t) = 1 - t^2 + \cos(t)$.

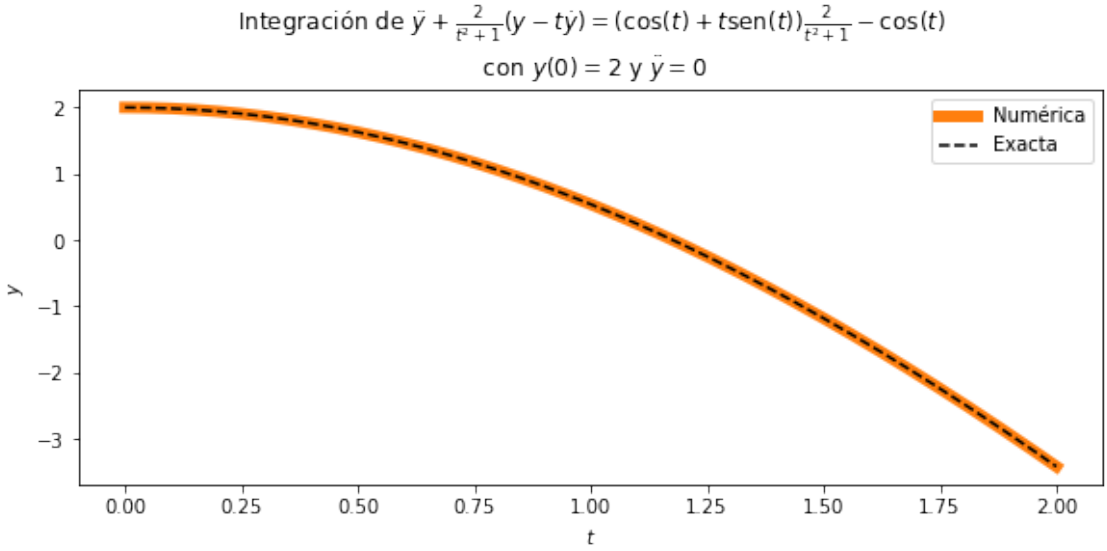
Notemos que llamando $u_0 = y$, $u_1 = \dot{y}$, podemos reducir esta EDO de segundo orden a un sistema de EDOs de primer orden de la siguiente manera

$$\begin{aligned} \dot{u}_0 &= u_1, & u_0(0) &= 2, \\ \dot{u}_1 &= -\frac{2}{t^2+1}(u_0 - tu_1) + \left(\cos(t) + t\mathrm{sen}(t)\right)\frac{2}{t^2+1} - \cos(t), & u_1(0) &= 0. \end{aligned}$$

Resolvamos este sistema usando un método de Runge-Kutta de orden 3 (RK3).

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

dt    = 1e-2                    # Paso temporal
tf    = 2                       # Tiempo final de integración
pasos = int(round(tf/dt))       # Cantidad de pasos

u = np.zeros( (pasos+1, 2) )  # Arreglo para u_0 (u[:,0]) y u_1 (u[:,1])

# Condiciones iniciales
u[0,0] = 2
u[0,1] = 0

# Variables para guardar los pasos intermedios de Runge-Kutta
R1 = np.zeros( 2 )
R2 = np.zeros( 2 )
R3 = np.zeros( 2 )

# Comienzo a integrar Runge-Kutta de 3er orden (RK3)
for n in range(0, pasos):
    # Primera etapa
    t1 = n*dt
    R1[0] = dt*u[n,1]
    R1[1] = -2*dt/(t1**2+1)*(u[n,0] - t1*u[n,1]) \
                + dt*((np.cos(t1) + t1*np.sin(t1))*2/(t1**2+1) - np.cos(t1) )

    # Segunda etapa
    t2 = t1 + dt/3
    R2[0] = dt*( u[n,1] + R1[1]/3 )
    R2[1] = -2*dt/(t2**2+1)*( u[n,0]+ R1[0]/3 - t2*(u[n,1]+R1[1]/3) ) \
                + dt*((np.cos(t2) + t2*np.sin(t2))*2/(t2**2+1) - np.cos(t2) )

    # Tercera etapa
    t3 = t1 + 2*dt/3
    R3[0] = dt*( u[n,1]+2*R2[1]/3 )
    R3[1] = -2*dt/(t3**2+1)*( u[n,0]+ 2*R2[0]/3 - t3*(u[n,1]+2*R2[1]/3)) \
                + dt*((np.cos(t3) + t3*np.sin(t3))*2/(t3**2+1) - np.cos(t3) )


    # Combino etapas
    u[n+1] = u[n] + (R1 + 3*R3)*1/4


# Grafico
t = np.arange(u.shape[0])*dt
fig, ax = plt.subplots(1, 1, figsize=(8,4), constrained_layout=True)
ax.plot(t, u[:,0], label="Numérica", lw=6, c="C1")
```

```
ax.plot(t, (1-t**2) + np.cos(t), "--k", label="Exacta")
ax.legend()
ax.set_title(r"Integración de $\ddot y + \frac{2}{t^2+1}(y-t\dot{y})= " +
             r"(\cos(t)+t\mathrm{sen}(t))\frac{2}{t^2+1}-\cos(t)$"+
             "\n con $y(0) = 2$ y $\ddot{y} = 0$")
ax.set_xlabel("$t$")
ax.set_ylabel("$y$");
```



1.2 Estabilidad

1.2.1 Regiones de estabilidad

Una cuestión muy importante que evitamos discutir hasta ahora es la convergencia de la solución numérica a la solución real.

Mencionamos en la práctica anterior que un requisito para que nuestro esquema sea convergente era que el método sea consistente. Esto quiere decir, que en el límite en que la grilla se vuelve infinitamente densa, el operador discreto aplicado a la solución real Y debe converger a la aplicación del operador continuo (lo vimos de esta manera en la práctica anterior, hay otras definiciones equivalentes).

Sin embargo, a la hora de realizar la integración temporal, esto no garantiza que la solución numérica sea convergente. Si inicialmente nuestra solución numérica contiene un pequeño error ϵ, es decir $y^0 = Y^0 + \epsilon$, ¿qué sucede con ϵ a medida que integramos en el tiempo?

Para responder esta pregunta es que analizamos la estabilidad. Solo veremos aquí algunas consideraciones generales sobre la estabilidad de distintos esquemas y no todas las maneras de estudiar la estabilidad de un integrador temporal, que es un tópico amplio.

Una manera de analizar la estabilidad es pregntarnos si el esquema temporal propuesto se mantiene

acotado al aplicarlo a la ecuación

$$\dot{y} = \lambda y,$$

donde $\lambda \in \mathbb{C}$. Noten la similitud con el análisis de Fourier para los errores que vieron en teóricas (estamos mirando el error de amplitud en vez del error de fase). Para $\mathrm{Re}(\lambda) < 0$, un esquema temporal apropiado debería devolver una solución acotada, sin embargo, veremos que esto no siempre es así.

En general, para un dado esquema temporal, habrá valores de Δt para los cuales la solución discreta de esta ecuación se mantenga acotada y otros para los cuales no. Estas serán las ***regiones de estabilidad*** e inestabilidad del método, respectivamente.

1.2.2 Ejemplos de regiones de estabilidad

Euler adelantado Como ejemplo podemos estudiar el método de Euler hacia adelante, y el análisis de estabilidad resulta

$$y^{n+1} = y^n + k\lambda y^n = (1 + k\lambda)y^n.$$

Si ahora lo escribimos en términos de la condición inicial vemos que

$$y^n = (1 + \bar{\lambda})^n y^0$$

con $\bar{\lambda} = k\lambda$, que se mantiene acotado solo sí $|1 + \bar{\lambda}| < 1$. Utilizamos la variable $\bar{\lambda}$ ya que para cualquier esquema acabaremos con un factor $\bar{\lambda} = k\lambda$ al discretizar la ecuación.

Obtuvimos entonces que la ***región de estabilidad*** para el método de Euler adelantado es la región $|1 + k\lambda| < 1$ (i.e. un círculo de radio 1 en el plano complejo centrado en $z = -1$).

Trapezoidal Si consideramos ahora el método trapezoidal tenemos

$$y^{n+1} = y^n + \frac{\bar{\lambda}}{2}(y^n + y^{n+1}) \qquad \Longrightarrow \qquad y^{n+1}\left(1 - \frac{\bar{\lambda}}{2}\right) = y^n\left(1 + \frac{\bar{\lambda}}{2}\right)$$

y por tanto

$$y^n = y_0 \left[\frac{1 + \frac{\bar{\lambda}}{2}}{1 - \frac{\bar{\lambda}}{2}}\right]^n .$$

La región de estabilidad estará dada entonces por $|1 + \bar{\lambda}/2| < |1 - \bar{\lambda}/2|$, que se satisface siempre que $\mathrm{Re}(\bar{\lambda}) < 0$, es decir, el método trapezoidal es ***incondicionalmente estable*** (es estable para todo k).

1.2.3 Rigidez de una EDO

La sección previa refirió solo a la estabilidad de EDOs lineales. Sin embargo, para el caso no-lineal, dada una solución a la ecuación a tiempo t^*, y^* (que obtuvimos, por ejemplo, numéricamente), podemos linealizar la ecuación alrededor de (t^*, y^*) y hacer un estudio de estabilidad lineal en un entorno de (t^*, y^*). Esta estrategia, que no utilizaremos en la materia, funciona aceptablemente en una gran cantidad de casos.

Luego de esta discusión, queda de manifiesto que **la estabilidad al integrar una EDO depende del esquema temporal escogido y de la propia EDO**. De esta observación surge el calificativo de **rígidas** (*stiff* en inglés) para ecuacióones diferenciales que requieren pasos temporales muy pequeños para mantener estable su integración. Una formulación alternativa de este concepto, es considerar rígidas a aquellas ecuaciones diferenciales que resultan extremadamente difíciles de integrar con métodos explícitos, que como veremos más adelante, suelen tener regiones de estabilidad más reducidas.

Aún cuando generalmente refiere ecuaciones diferenciales, y así lo usaremos en la materia, el calificativo de rígido es más apropiado para referirse al problema de valores iniciales en su conjunto. Por ejemplo, una EDO puede ser más o menos estable para integrar en función de su condición inicial (i.e. de la solución particular), o del intervalo donde estamos buscando una solución.

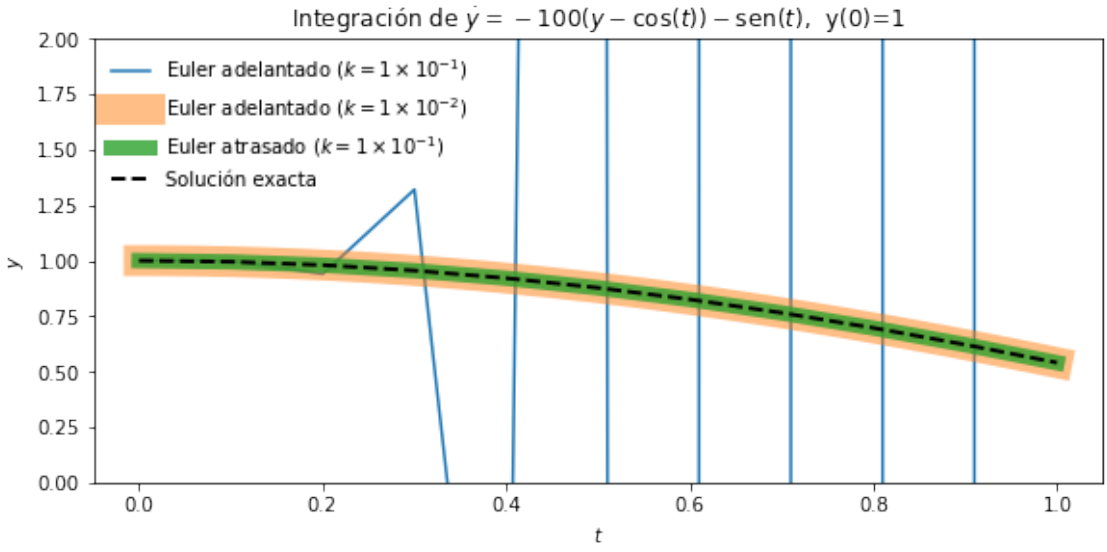
Para ver los conceptos de rigidez y estabilidad, consideremos el ejemplo

$$\dot{y}(t) = -100\,[y(t) - \cos(t)] - \operatorname{sen}(t), \qquad y(0) = 1,$$

para $0 \leq t \leq 1$. Este problema de valores iniciales tiene como solución $y(t) = \cos(t)$, y se anula el término entre corchetes. Veamos que sucede numéricamente

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

dt = 1e-1  # Paso temporal
y0 = 1     # Condición inicial

tf    = 1                  # Tiempo final de integración
pasos = int(round(tf/dt))  # Cantidad de pasos

# Inicializo y para los dos integradores que voy a usar
y_ex1 = np.zeros( pasos+1 )
y_im = np.zeros( pasos+1 )

# Escribo la condición inicial
y_ex1[0] = y0
y_im[0] = y0

# Integro con Euler adelantado (explícito) y atrasado (implícito)
for n in range(0, pasos):
    t_ex =     n*dt  # Tiempo actual
    t_im = (n+1)*dt  # Tiempo siguiente

    # Euler adelantado
    y_ex1[n+1] = y_ex1[n] + dt*(-100*(y_ex1[n]-np.cos(t_ex)) - np.sin(t_ex))

    # Euler atrasado, como la EDO es lineal, puedo despejar y^{n+1}
    y_im[n+1] = (y_im[n] + dt*(100*np.cos(t_im) - np.sin(t_im))) / (1+100*dt)

# Pruebo ahora reducir el paso en un factor 10
dt2      = dt/10
```

```
pasos2   = int(round(tf/dt2))
y_ex2    = np.zeros( pasos2+1 )
y_ex2[0] = y0
for n in range(0, pasos2):
    t_ex = n*dt2
    # Euler adelantado
    y_ex2[n+1] = y_ex2[n] + dt2*(-100*(y_ex2[n]-np.cos(t_ex)) - np.sin(t_ex))

# Grafico
t1 = np.arange(0, y_ex1.size)*dt
t2 = np.arange(0, y_ex2.size)*dt2

fig, ax = plt.subplots(1, figsize=(8,4), constrained_layout=True)
ax.plot(t1, y_ex1, label=r"Euler adelantado ($k=1\times10^{-1}$)")
ax.plot(t2, y_ex2, label=r"Euler adelantado ($k=1\times10^{-2}$)", lw=16,
        alpha=.5)
ax.plot(t1, y_im, label=r"Euler atrasado ($k=1\times10^{-1}$)", lw=8, alpha=0.8)
ax.plot(t1, np.cos(t1), '--k', label="Solución exacta", lw=2)
ax.legend(frameon=False, loc="upper left")
ax.set_title("Integración de $\dot y = -100(y-\cos(t)) - \mathrm{sen}(t)$," +
             "  y(0)=1")
ax.set_xlabel("$t$")
ax.set_ylabel("$y$")
ax.set_ylim(0, 2);
```



Vemos que el método explícito diverge para $k = 1 \times 10^{-1}$, mientras que el implícito se mantiene acotado y además converge a la solución exacta. Esto puede verse a partir de lo que vimos en la sección previa, ya que linealizando la ecuación tenemos $\lambda = -100$ y $\bar{\lambda} = k\lambda = -10$ está fuera de la

región de convergencia del método de Euler adelantado.

Alternativamente, podemos comprender este fenómeno de la siguiente manera. Reparemos en que para una condición inicial ligeramente diferente el término entre corchetes resulta dominante en los instantes iniciales, convergiendo en una escala temporal de $0,01$ $(1/100)$ a una solución muy similar a $\cos(t)$. Sin embargo, el método explícito no logra capturar esta escala temporal y acaba por diverger.

Es por ello que un formulación igualmente válida de rigidez, pero más interesante para este curso será la siguiente:

Una EDO rígida es aquella que involucra escalas muy diversas.

Vean que, en nuestro caso, el problema que nos presenta la EDO propuesta es que nos interesa el comportamiento para tiempos $\mathcal{O}(1)$ (el orden del período de $\cos(t)$); sin embargo, para alcanzar una solución aceptable de este comportamiento de baja frecuencia, debemos resolver también las escalas $\mathcal{O}(0,01)$. Pensado físicamente, ese es el motivo por el cual nuestro integrador explícito presenta dificultades, requiriendo 10 veces más pasos para converger a la solución correcta.

Espero que con esto también resulte más clara la importancia de los métodos implícitos que, para el caso no-lineal, pueden requerir un trabajo considerablemente mayor por iteración, con respecto a los explícitos. Sin embargo, generalmente se mantienen estables tomando pasos temporales más grandes. Veremos en el resto de la materia que la elección de un esquema temporal explícito o implícito será problema-dependiente.

1.2.4 Resumen de regiones de estabilidad

Les dejamos a continuación diagramas que esquematizan las regiones de estabilidad de algunas familias de esquemas temporales. En todos los casos los diagramas se hallan en unidades de $\bar{\lambda}$ (noten las diferencias en los límites de cada panel).

Fuera de esta gráfica quedó el método *Salto de rana*, que solo es estable en la región $\mathrm{Re}(\bar{\lambda}) = 0 \ \wedge \ |\mathrm{Im}(\bar{\lambda})| \leq 1$, siendo apropiado solo para problemas de oscilaciones sin amortiguamiento.

Para los métodos explícitos (Adams-Bashforth y Runge-Kutta) y para Adams-Moulton de orden 3 en adelante, las regiones de estabilidad son las regiones interiores a cada curva (sombreadas en los gráficos). Para Adams-Moulton de orden 1 la región de estabilidad es la exterior a la circunsferencia, mientras que para orden 2 es todo el semiplano $\mathrm{Re}(\bar{\lambda}) \leq 0$. En el caso de diferenciación hacia atrás, las regiones de estabilidad son exteriores a cada una de las curvas.

Recuerden que coinciden los métodos de Runge-Kutta y Adams-Bashforth de orden 1, siendo ambos el método de Euler adelantado. Por su parte, los esquemas de Adams-Moulton de orden 1 y 2 representan a Euler atrasado y a la regla trapezoidal, respectivamente.

Vale notar también que en los métodos multipaso las regiones de estabilidad van disminuyendo a medida que aumenta el orden, mientras que con Runge-Kutta (hasta orden 4) pasa lo contrario.

1.2.5 Relación entre convergencia, consistencia y estabilidad

La importancia de la estabilidad queda resumida en el teorema de Dahlquist[†], que enuncia que para un esquema numérico que aproxime a una EDO

$$\text{Convergente} \quad \Longleftrightarrow \quad \text{Consistente} \wedge \text{Estable} \qquad \text{(T. de Dahlquist)}$$

1.3 Ejemplo integrador: Ecuación de Lane-Emden

Consideremos ahora el siguiente problema de valores iniciales no lineal

$$t^2\ddot{y} + 2t\dot{y} + t^2 y^\gamma = 0, \qquad y(0) = 1, \qquad \dot{y}(0) = 0.$$

Esta ecuación es conocida como ecuación de Lane-Emden y describe (adimensionalmente) la presión y la densidad en función de la distancia para una esfera de fluido autogravitante en equilibrio hidrostático, bajo la aproximación politrópica.

Llamando $u_0 = y$, $u_1 = \dot{y}$, podemos reducir esta EDO de segundo orden a un sistema de EDOs de primer orden de la siguiente manera

$$\begin{aligned} \dot{u}_0 &= u_1, & u_0(0) &= 1, \\ \dot{u}_1 &= -u_0^\gamma - \frac{2u_1}{t} & u_1(0) &= 0. \end{aligned}$$

Noten que para integrar vamos a tener una singularidad al calcular ui_1 en $t = 0$. Puede mostrarse que, para todo valor de γ, las soluciones a la ecuación de Lane-Emden verifican $\ddot{y}(0) = -y(0)^\gamma/3$. Vamos a aprovechar este resultado para conservar el orden del integrador. Vale remarcar que si no repararamos en este hecho, podríamos integrar la ecuación de cualquier modo, pero los integradores de alto orden temporal verían su orden de precisión disminuido.

Vamos a resolver este sistema utilizando un método de Adams-Moulton de 3er orden (i.e., de 2 pasos). Para ello, tendremos que conseguir de alguna manera $\mathbf{u}^1$ (la solución para el primer paso temporal) y a partir de allí podremos aplicar

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \frac{k}{12}\left(5\mathbf{f}^{n+1} + 8\mathbf{f}^n - \mathbf{f}^{n-1}\right).$$

Esta ecuación implícita para $\mathbf{u}^{n+1}$ (aparece en $\mathbf{f}^{n+1}$ y explícitamente en el miembro izquierdo), la resolveremos mediante un método iterativo de Newton-Krylov (en particular, LGMRES) que buscará las raíces de la función

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} - \mathbf{u}^n - \frac{k}{12}\left(5\mathbf{f}(\mathbf{x}) + 8\mathbf{f}^n - \mathbf{f}^{n-1}\right),$$

donde $\mathbf{f}(t^{n+1}, \mathbf{x})$ es el sistema de Lane-Emden evaluado en $(t^{n+1}, \mathbf{x})$. Vale remarcar que no es necesario que como funcionan los algoritmos de Newton-Krylov, solo precisan saber que permiten obtener raices de forma eficiente y que podemos acceder a él mediante una biblioteca preinstalalada en Colab, SciPy, a través de `scipy.optimize.newton_krylov`.

Para obtener $\mathbf{u}^1$ vamos a usar otro método de la familia de Adams-Moulton, en particular un método de segundo orden. Para ello utilizaremos la misma idea de búsqueda de raices que mencionamos más arriba pero con el esquema

$$\mathbf{u}^1 = \mathbf{u}^0 + \frac{k}{2}(\mathbf{f}^1 + \mathbf{f}^0).$$

Sin embargo, este método genera $\mathbf{u}^1$ con un orden de aproximación menor que el que tendrá el integrador que usaremos en adelante (2 y 3 respectivamente). Para mantener el orden de aproximación tendremos que usar a la hora de inicializar un paso temporal k' más chico que el que usaremos

luego para continuar la integración, k. Este k' deberá verificar ser el mínimo entero mayor que $k^{1-3/2}$ (3 y 2 vienen del orden de integración de cada método), es decir

$$k' = \lceil k^{1-\frac{3}{2}} \rceil,$$

con $\lceil\rceil$ la función techo.

El siguiente código define una función que nos permite inicializar la cantidad de pasos deseada para la ecuación de Lane-Emden. También permite ser utilizada para inicializar métodos de otro orden.

[ ]:
```
def inicializar(cond_inicial, gamma, dt, pasos, orden):
    """ Inicializa Lane-Emden usando Adams-Moulton de 2do orden (trapezoidal)
    utilizando un paso de forma tal de mantener consistente el orden de
    aproximación posterior:
        Entrada:
            - cond_incial: arreglo de dimensiones (2) con las condiciones
                iniciales para u0 y u1.
            - gamma: índice politrópico.
            - dt: paso temporal del método que continuará la integración;
            - pasos: cantidad de pasos a generar usando AM2;
            - orden: orden del método que continuará la integración.
        Devuelve:
            - u: arreglo de dimensiones (pasos, 2) con la ecuación de Lane-Emden
            integrada sobre la cantidad indicada de pasos."""
    import numpy as np
    import scipy.optimize as spoptimize

    # Calculo los pasos para el inicializador por cada paso que debe realizar
    # el integrador de orden superior.
    pasos_ini = dt**(1-orden/2)
    #  pasos_ini seguramente no sea entero. Tomo el primer entero mayor a
    # pasos_ini y defino el dt del inicializador coherentemente
    pasos_ini = int(np.ceil(pasos_ini))
    dt_ini    = dt/pasos_ini

    # Creo un arreglo para ir guardando la integración y agrego condicion␣
↪inicial
    u_ini    = np.zeros( (pasos_ini*pasos+1, 2) )
    u_ini[0] = cond_inicial

    fn   = np.zeros( 2 )  # Donde voy a ir guardando f^n
    fn1  = np.zeros( 2 )  # Donde voy a ir guardando f^{n-1}

    # Comienzo a integrar usando una regla trapezoidal implícita (AM2)
    for n in range(0, pasos_ini*pasos):
        ts  = (n+1)*dt_ini  # t^{n+1}
        tn  =     n*dt_ini  # t^n

        fn[0]  =  u_ini[n,1]                          # f_1^n
```

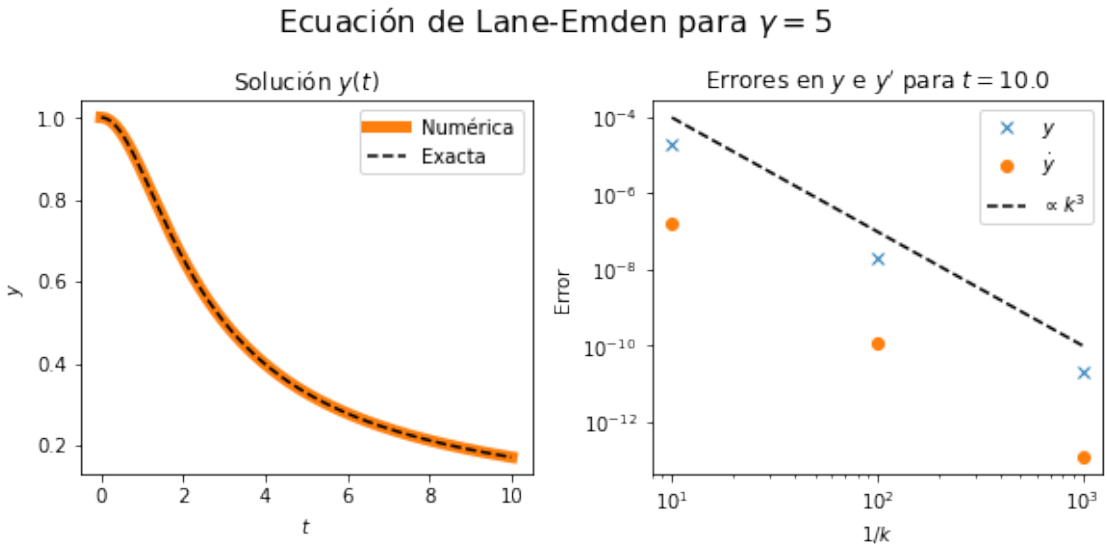
```
        fn[1]  = -u_ini[n,0]**gamma - 2*u_ini[n,1]/tn  # f_2^n

        # Salvo la singularidad en 0 para y''.
        if n==0: fn[1] = -1/3

        # Obtengo estimación incial para buscar raices de y^{n+1} usando Euler
        est    = u_ini[n] + dt_ini*fn

        # Función a la que le voy a buscar las raices
        def f_raices(us):
            fs = np.array([ us[1], -us[0]**gamma - 2*us[1]/ts ])  # f^{n+1}
            return us - u_ini[n] - (fs + fn)*dt_ini/2             # AM2

        # Le busco las raices usando Newton-Krylov (i.e. obtengo u_ini^{n+1})
        u_ini[n+1] = spoptimize.newton_krylov(f_raices, est)

    # Devuelvo solo los valores de u para el paso temporal original
    return u_ini[pasos_ini::pasos_ini]
```

Luego de correr esa celda, ya pueden correr el siguiente código para integrar la ecuación de Lane-Emden con el mecanismo propuesto.

[ ]:
```
# Vamos a integrar Lane-Emden con Adams-Moulton de 3er orden (2 pasos)
import numpy as np
import scipy.optimize as spoptimize
import matplotlib.pyplot as plt

# Las siguientes dos líneas filtran algunas advertencias que genera el método
# de búsqueda de raices. No las usen en sus códigos.
import warnings
warnings.filterwarnings('ignore')


# Para algunos valores de gamma conozco la solución analítica, la aprovecho.
def sol_analitica(t, gamma):
    if gamma == 0:
        return np.array([ 1-t**2/6, -t/3 ])
    if gamma == 1:
        return np.array([ np.sin(t)/t, (t*np.cos(t) - np.sin(t))/t**2 ])
    if gamma == 5:
        return np.array([ 1/np.sqrt(1+t**2/3), -np.sqrt(3)*t/(t**2+3)**1.5 ])


tf    = 10                              # "Tiempo" final de integración
gamma = 5                               # Índice politrópico
analitica = gamma in [0, 1, 5]          # Veo si existe sol. analitica para gamma
dts   = np.array([ 1e-1, 1e-2, 1e-3 ]) # Pasos temporales a explorar
```

```
# Si no tengo solución analitica elijo un único dt para integrar.
if not analitica: dt = np.array([1e-3])

fn   = np.zeros( 2 )  # Donde voy a ir guardando f^n
fn1  = np.zeros( 2 )  # Donde voy a ir guardando f^{n-1}

# Si hay solución analítica voy a ir guardando el error de cada integración
if analitica: errs = np.zeros( (dts.size, 2) )

# Integro para cada dt
for i, dt in enumerate(dts):
    pasos = int(round(tf/dt))     # Cantidad de pasos
    u = np.zeros( (pasos+1, 2) )  # Arreglo para u_0 (u[:,0]) y u_1 (u[:,1])

    # Condiciones iniciales
    u[0,0] = 1
    u[0,1] = 0

    # Inicializo el paso que no puedo hacer con AM3.
    u[1] = inicializar(u[0], gamma, dt, 1, 3)

    # Comienzo a integrar Adams-Moulton de 3er orden (AM3)
    for n in range(1, pasos):
        ts  = (n+1)*dt  # t^{n+1}
        tn  =     n*dt  # t^n
        tn1 = (n-1)*dt  # t^{n-1}

        if n == 1: tn1 = 1e-50  # Para evitar 0/0 al evaluar f en la t=0.

        fn[0]  =  u[n,1]                              # f_1^n
        fn[1]  = -u[n,0]**gamma - 2*u[n,1]/tn         # f_2^n

        fn1[0] =  u[n-1,1]                            # f_1^{n-1}
        fn1[1] = -u[n-1,0]**gamma - 2*u[n-1,1]/tn1    # f_2^{n-1}

        # Salvo la singularidad para y'' en t=0
        if n==1: fn1[1] = -1/3

        # Obtengo estimación incial para buscar raices de y^{n+1} usando Euler
        est    = u[n] + dt*fn

        # Función a la que le voy a buscar las raices
        def f_raices(us):
            fs = np.array([ us[1], -us[0]**gamma - 2*us[1]/ts ])  # f^{n+1}
            return us - u[n] - ( 5*fs + 8*fn - fn1)*dt/12         # AM3
```

```
        # Le busco las raices con Newton-Krylov (i.e. obtengo u^{n+1})
        u[n+1] = spoptimize.newton_krylov(f_raices, est)

    # Si tengo solución analítica, calculo el error
    if analitica: errs[i] = np.abs(u[n+1] - sol_analitica(ts, gamma))

# Grafico
if analitica:
    fig, axs = plt.subplots(1, 2, figsize=(8,4), constrained_layout=True)
else:
    fig, ax = plt.subplots(1, 1, figsize=(4,4), constrained_layout=True)
    axs = [ax]

t = np.arange(u.shape[0])*dt
axs[0].plot(t, u[:,0], c="C1", label="Numérica", lw=6)
axs[0].set_xlabel("$t$")
axs[0].set_ylabel("$y$")
fig.suptitle(f"Ecuación de Lane-Emden para $\gamma={gamma}$", fontsize=16)
axs[0].set_title("Solución $y(t)$")
if analitica:
    axs[0].plot(t, sol_analitica(t,gamma)[0], "--k", label="Exacta")
    axs[1].loglog(1/dts, errs[:,0], 'x', label="$y$")
    axs[1].loglog(1/dts, errs[:,1], 'o', label="$\dot{y}$")
    axs[1].loglog(1/dts, 1e-1*dts**3, "--k", label="$\propto k^3$")
    axs[1].legend()
    axs[1].set_title(f"Errores en $y$ e $y'$ para $t={t[-1]:.1f}$")
    axs[1].set_xlabel("$1/k$")
    axs[1].set_ylabel("Error")
axs[0].legend();
```



Como pueden probar ustedes mismos, el código propuesto aprovecha el hecho de que para $\gamma \in \{0, 1, 5\}$ existe una solución analítica y calcula el error en la integración. Pueden ver que para estos valores de γ el orden de precisión del método es el esperado (i.e. $\propto k^3$), excepto para $\gamma = 0$ donde en todos los casos está cerca del error asociado a la precisión aritmética finita.

Vale remarcar finalmente que el uso de un método implícito en este ejemplo conlleva un objetivo puramente pedagógico, de forma de ilustrar su uso para un caso no-lineal y de resolver el problema de la inicialización. Para una ecuación poco rígida como esta resulta más eficiente el uso de algoritmos explícitos, como RK4.

1.4 Paso adaptativo

Un elemento que no queríamos dejar de mencionar, aunque no lo veremos en el curso, es que los métodos vistos anteriormente pueden reformularse, con mayor o menor dificultad, para funcionar con k variable. De esta manera, cuando la EDO se comporta más rígidamente se disminuye k, mientras que si luego la integración se vuelve menos rígida es posible agrandar el paso. Numerosos solvers automáticos de EDOs, como `scipy.integrate.odeint` implementan esta estrategia.

Sin embargo, esta estrategia no siempre resulta computacionalmente óptima cuando se tiene conocimiento del problema físico en cuestión. Por ejemplo, al integrar las ecuaciones de Navier-Stokes, se conoce a priori el rango de escalas temporales que se deben resolver correctamente y, por tanto, las estrategias con k fijo pueden resultar más apropiadas.

1.5 Referencias:

- *Finite Difference and Spectral Methods for Ordinary and Partial Differential Equations*; N. L. Trefethen (1996).
- *An Introduction to Numerical Modeling of the Atmosphere*; D. A. Randall.