Simulación de Sistemas Continuos y a Tramos

Prof. Dr. François E. Cellier
Institut für Computational Science
ETH Zürich

25 de junio 2007

Análisis del Error por Truncamiento

Queremos hacer un *análisis del error por truncamiento* del método explícito ya encontrado antes con una predicción FE y con una sola corrección BE:

$$\begin{aligned} \text{predicción:} \quad & \dot{\mathbf{x}}_\mathbf{k} = \mathbf{f}(\mathbf{x}_\mathbf{k}, t_k) \\ & \mathbf{x}^\mathbf{P}_{\mathbf{k+1}} = \mathbf{x}_\mathbf{k} + h \cdot \dot{\mathbf{x}}_\mathbf{k} \end{aligned}$$

$$\begin{aligned} \text{corrección:} \quad & \dot{\mathbf{x}}^\mathbf{P}_{\mathbf{k+1}} = \mathbf{f}(\mathbf{x}^\mathbf{P}_{\mathbf{k+1}}, t_{k+1}) \\ & \mathbf{x}^\mathbf{C}_{\mathbf{k+1}} = \mathbf{x}_\mathbf{k} + h \cdot \dot{\mathbf{x}}^\mathbf{P}_{\mathbf{k+1}} \end{aligned}$$

Obtenemos:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_k} + h \cdot \mathbf{f_k}, t_k + h)$$

Queremos desarrollar la expresión no lineal en una serie de Taylor multidimensional:

$$f(x + \Delta x, y + \Delta y) \approx f(x, y) + \frac{\partial f(x, y)}{\partial x} \cdot \Delta x + \frac{\partial f(x, y)}{\partial y} \cdot \Delta y$$

Entonces:

$$\mathbf{f}(\mathbf{x_k} + h \cdot \mathbf{f_k}, t_k + h) \approx \mathbf{f}(\mathbf{x_k}, t_k) + \frac{\partial \mathbf{f}(\mathbf{x_k}, t_k)}{\partial \mathbf{x}} \cdot (h \cdot \mathbf{f_k}) + \frac{\partial \mathbf{f}(\mathbf{x_k}, t_k)}{\partial t} \cdot h$$

Análisis del Error por Truncamiento II

Se obtiene la aproximación:

$$\mathbf{x_{k+1}} \approx \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_k}, t_k) + h^2 \cdot \left(\frac{\partial \mathbf{f}(\mathbf{x_k}, t_k)}{\partial \mathbf{x}} \cdot \mathbf{f_k} + \frac{\partial \mathbf{f}(\mathbf{x_k}, t_k)}{\partial t}\right)$$

Queremos comparar esa aproximación con la serie de Taylor truncado después del término cuadrado:

$$\mathbf{x_{k+1}} \approx \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_k}, t_k) + \frac{h^2}{2} \cdot \dot{\mathbf{f}}(\mathbf{x_k}, t_k)$$

donde:

$$\dot{\mathbf{f}}(\mathbf{x_k}, t_k) = \frac{d\mathbf{f}(\mathbf{x_k}, t_k)}{dt} = \frac{\partial \mathbf{f}(\mathbf{x_k}, t_k)}{\partial \mathbf{x}} \cdot \frac{d\mathbf{x_k}}{dt} + \frac{\partial \mathbf{f}(\mathbf{x_k}, t_k)}{\partial t}$$

y:

$$\frac{d\mathbf{x_k}}{dt} = \dot{\mathbf{x}}_\mathbf{k} = \mathbf{f_k}$$

Entonces:

$$\mathbf{x_{PC}}(k+1) \approx \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_k}, t_k) + h^2 \cdot \dot{\mathbf{f}}(\mathbf{x_k}, t_k)$$

El Algoritmo de Heun

Comparando las dos aproximaciones de FE y de PC:

$$\mathbf{x_{FE}}(k+1) \approx \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_k}, t_k)$$
$$\mathbf{x_{PC}}(k+1) \approx \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{x_k}, t_k) + h^2 \cdot \dot{\mathbf{f}}(\mathbf{x_k}, t_k)$$

puede verse que esas dos aproximaciones pueden combinarse fácilmente con la tarea de obtener una aproximación de segundo orden:

$$\mathbf{x}(k+1) = 0.5 \cdot (\mathbf{x_{PC}}(k+1) + \mathbf{x_{FE}}(k+1))$$

es decir:

predicción:
$$\dot{\mathbf{x}}_\mathbf{k} = \mathbf{f}(\mathbf{x_k}, t_k)$$
$$\mathbf{x}^\mathbf{P}_{\mathbf{k+1}} = \mathbf{x_k} + h \cdot \dot{\mathbf{x}}_\mathbf{k}$$

corrección:
$$\dot{\mathbf{x}}^\mathbf{P}_{\mathbf{k+1}} = \mathbf{f}(\mathbf{x}^\mathbf{P}_{\mathbf{k+1}}, t_{k+1})$$
$$\mathbf{x}^\mathbf{C}_{\mathbf{k+1}} = \mathbf{x_k} + 0.5 \cdot h \cdot (\dot{\mathbf{x}}_\mathbf{k} + \dot{\mathbf{x}}^\mathbf{P}_{\mathbf{k+1}})$$

Ese método se llama el *algoritmo de Heun*.

Métodos Runge Kutta Explícitos de Segundo Orden

Podemos investigar si es posible combinar más de dos aproximaciones diferentes con la tarea de obtener *algoritmos de órdenes más altos*.

Empezamos con una sola correción como antes, pero esta vez parametrizada:

$$\begin{aligned} \text{predicción:} \quad & \dot{\mathbf{x}}_\mathbf{k} = \mathbf{f}(\mathbf{x}_\mathbf{k}, t_k) \\ & \mathbf{x}^\mathbf{P} = \mathbf{x}_\mathbf{k} + h \cdot \beta_{11} \cdot \dot{\mathbf{x}}_\mathbf{k} \end{aligned}$$

$$\begin{aligned} \text{corrección:} \quad & \dot{\mathbf{x}}^\mathbf{P} = \mathbf{f}(\mathbf{x}^\mathbf{P}, t_k + \alpha_1 \cdot h) \\ & \mathbf{x}^\mathbf{C}_{\mathbf{k+1}} = \mathbf{x}_\mathbf{k} + h \cdot (\beta_{21} \cdot \dot{\mathbf{x}}_\mathbf{k} + \beta_{22} \cdot \dot{\mathbf{x}}^\mathbf{P}) \end{aligned}$$

Desarrollando en una serie de Taylor como antes se obtiene:

$$\mathbf{x}^\mathbf{C}_{\mathbf{k+1}} = \mathbf{x}_\mathbf{k} + h \cdot (\beta_{21} + \beta_{22}) \cdot \mathbf{f}_\mathbf{k} + \frac{h^2}{2} \cdot [2 \cdot \beta_{11} \cdot \beta_{22} \cdot \frac{\partial \mathbf{f}_\mathbf{k}}{\partial \mathbf{x}} \cdot \mathbf{f}_\mathbf{k} + 2 \cdot \alpha_1 \cdot \beta_{22} \cdot \frac{\partial \mathbf{f}_\mathbf{k}}{\partial t}]$$

Métodos Runge Kutta Explícitos de Segundo Orden II

Esta aproximación:

$$\mathbf{x}^{\mathbf{C}}_{\mathbf{k+1}} = \mathbf{x_k} \; + \; h \cdot (\beta_{21} + \beta_{22}) \cdot \mathbf{f_k} \; + \; \frac{h^2}{2} \cdot [2 \cdot \beta_{11} \cdot \beta_{22} \cdot \frac{\partial \mathbf{f_k}}{\partial \mathbf{x}} \cdot \mathbf{f_k} \; + \; 2 \cdot \alpha_1 \cdot \beta_{22} \cdot \frac{\partial \mathbf{f_k}}{\partial t}]$$

puede compararse con la expansión de Taylor cortado después del término de segundo orden:

$$\mathbf{x_{k+1}} \approx \mathbf{x_k} \; + \; h \cdot \mathbf{f_k} \; + \; \frac{h^2}{2} \cdot [\frac{\partial \mathbf{f_k}}{\partial \mathbf{x}} \cdot \mathbf{f_k} \; + \; \frac{\partial \mathbf{f_k}}{\partial t}]$$

De esa forma obtenemos condiciones generales que nos garantizan *algoritmos de segundo orden*.

$$\begin{aligned} \beta_{21} + \beta_{22} &= 1 \\ 2 \cdot \alpha_1 \cdot \beta_{22} &= 1 \\ 2 \cdot \beta_{11} \cdot \beta_{22} &= 1 \end{aligned}$$

Métodos Runge Kutta Explícitos de Segundo Orden III

Es cierto que el *algoritmo de Heun* con:

$$\alpha = \begin{pmatrix} 1 \\ 1 \end{pmatrix}; \quad \beta = \begin{pmatrix} 1 & 0 \\ 0.5 & 0.5 \end{pmatrix}$$

satisface las tres ecuaciones no lineales en cuatro incógnitas. α_2 es el tiempo relativo de la corrección que siempre debe ser 1 (al final del paso debemos llegar a $t^* + h$).

A veces, los métodos Runge Kutta también se especifican usando una *tabla de Butcher*:

0	0	0
1	1	0
x	1/2	1/2

donde la primera fila representa la evaluación de la diferenciación en el tiempo $t*$, la segunda fila denota la predicción, es decir la primera etapa del algoritmo, mientras la última fila indica la corrección, es decir la aproximación del valor del vector de variables del estado en el tiempo $t^* + h$. La columna a la izquierda marca los instantes en el tiempo de cada etapa mientras las demás columnas dan los valores de los parámetros de cada etapa.

La Regla Expícita del Punto Medio

Otro algoritmo que satisface las tres ecuaciones es caracterizado por:

$$\alpha = \begin{pmatrix} 0.5 \\ 1 \end{pmatrix}; \quad \beta = \begin{pmatrix} 0.5 & 0 \\ 0 & 1 \end{pmatrix}$$

es decir por la tabla de Butcher:

0	0	0
1/2	1/2	0
x	0	1

Ese algoritmo se llama la *regla expícita del punto medio*.

El algoritmo puede implementarse de la manera siguiente:

predicción:
$$\begin{aligned} \dot{\mathbf{x}}_{\mathbf{k}} &= \mathbf{f}(\mathbf{x}_{\mathbf{k}}, t_k) \\ \mathbf{x}^{\mathbf{P}}_{\mathbf{k}+\frac{1}{2}} &= \mathbf{x}_{\mathbf{k}} + \tfrac{h}{2} \cdot \dot{\mathbf{x}}_{\mathbf{k}} \end{aligned}$$

corrección:
$$\begin{aligned} \dot{\mathbf{x}}^{\mathbf{P}}_{\mathbf{k}+\frac{1}{2}} &= \mathbf{f}(\mathbf{x}^{\mathbf{P}}_{\mathbf{k}+\frac{1}{2}}, t_{k+\frac{1}{2}}) \\ \mathbf{x}^{\mathbf{C}}_{\mathbf{k}+1} &= \mathbf{x}_{\mathbf{k}} + h \cdot \dot{\mathbf{x}}^{\mathbf{P}}_{\mathbf{k}+\frac{1}{2}} \end{aligned}$$

Este método es un poco más económico que el algoritmo de Heun, porque su tabla de Butcher contiene un cero más.

La Familia de los Métodos Runge Kutta Explícitos

Si permitimos más predicciones (etapas), pueden obtenerse *métodos de órdenes más altos*:

etapa 0:
$$\dot{\mathbf{x}}^{\mathbf{P_0}} = \mathbf{f}(\mathbf{x_k}, t_k)$$

etapa j:
$$\mathbf{x}^{\mathbf{P_j}} = \mathbf{x_k} + h \cdot \sum_{i=1}^{j} \beta_{ji} \cdot \dot{\mathbf{x}}^{\mathbf{P_{i-1}}}$$
$$\dot{\mathbf{x}}^{\mathbf{P_j}} = \mathbf{f}(\mathbf{x}^{\mathbf{P_j}}, t_k + \alpha_j \cdot h)$$

última etapa:
$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot \sum_{i=1}^{\ell} \beta_{\ell i} \cdot \dot{\mathbf{x}}^{\mathbf{P_{i-1}}}$$

El más conocido de esa clase de algoritmos es un *método Runge Kutta explícito de cuarto orden (RK4)* caracterizado por:

$$\alpha = \begin{pmatrix} 1/2 \\ 1/2 \\ 1 \\ 1 \end{pmatrix} ; \quad \beta = \begin{pmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/6 & 1/3 & 1/3 & 1/6 \end{pmatrix}$$

El Algoritmo RK4

Entonces:

0	0	0	0	0
1/2	1/2	0	0	0
1/2	0	1/2	0	0
1	0	0	1	0
x	1/6	1/3	1/3	1/6

El *algoritmo RK4* puede implementarse de la forma siguiente:

etapa 0: $\dot{\mathbf{x}}_\mathbf{k} = \mathbf{f}(\mathbf{x}_\mathbf{k}, t_k)$

etapa 1:
$$\mathbf{x}^{\mathbf{P1}} = \mathbf{x}_\mathbf{k} + \frac{h}{2} \cdot \dot{\mathbf{x}}_\mathbf{k}$$
$$\dot{\mathbf{x}}^{\mathbf{P1}} = \mathbf{f}(\mathbf{x}^{\mathbf{P1}}, t_{k+\frac{1}{2}})$$

etapa 2:
$$\mathbf{x}^{\mathbf{P2}} = \mathbf{x}_\mathbf{k} + \frac{h}{2} \cdot \dot{\mathbf{x}}^{\mathbf{P1}}$$
$$\dot{\mathbf{x}}^{\mathbf{P2}} = \mathbf{f}(\mathbf{x}^{\mathbf{P2}}, t_{k+\frac{1}{2}})$$

etapa 3:
$$\mathbf{x}^{\mathbf{P3}} = \mathbf{x}_\mathbf{k} + h \cdot \dot{\mathbf{x}}^{\mathbf{P2}}$$
$$\dot{\mathbf{x}}^{\mathbf{P3}} = \mathbf{f}(\mathbf{x}^{\mathbf{P3}}, t_{k+1})$$

etapa 4: $\mathbf{x}_{\mathbf{k+1}} = \mathbf{x}_\mathbf{k} + \frac{h}{6} \cdot [\dot{\mathbf{x}}_\mathbf{k} + 2 \cdot \dot{\mathbf{x}}^{\mathbf{P1}} + 2 \cdot \dot{\mathbf{x}}^{\mathbf{P2}} + \dot{\mathbf{x}}^{\mathbf{P3}}]$

Se trata de un *algoritmo explícito en un paso de cuarto orden en cuatro etapas*.

La Historia de los Métodos Runge Kutta Explícitos

Encontrado por	Año	Orden	Etapas
Euler	1768	1	1
Runge	1895	4	4
Heun	1900	2	2
Kutta	1901	5	6
Huťa	1956	6	8
Shanks	1966	7	9
Curtis	1970	8	11

Table: Historia de los métodos Runge Kutta explícitos

Dominios de la Estabilidad Numérica de Algoritmos RK

Empezamos aplicando el algoritmo de Heun a un sistema lineal:

$$\begin{array}{lll} \text{predicción:} & \dot{\mathbf{x}}_{\mathbf{k}} = \mathbf{A} \cdot \mathbf{x}_{\mathbf{k}} \\ & \mathbf{x}^{\mathbf{P}}_{\mathbf{k+1}} = \mathbf{x}_{\mathbf{k}} + h \cdot \dot{\mathbf{x}}_{\mathbf{k}} \\ \text{corrección:} & \dot{\mathbf{x}}^{\mathbf{P}}_{\mathbf{k+1}} = \mathbf{A} \cdot \mathbf{x}^{\mathbf{P}}_{\mathbf{k+1}} \\ & \mathbf{x}^{\mathbf{C}}_{\mathbf{k+1}} = \mathbf{x}_{\mathbf{k}} + 0.5 \cdot h \cdot (\dot{\mathbf{x}}_{\mathbf{k}} + \dot{\mathbf{x}}^{\mathbf{P}}_{\mathbf{k+1}}) \end{array}$$

Es decir:

$$\mathbf{x}^{\mathbf{C}}_{\mathbf{k+1}} = [\mathbf{I}^{(\mathbf{n})} + \mathbf{A} \cdot h + \frac{(\mathbf{A} \cdot h)^2}{2}] \cdot \mathbf{x}_{\mathbf{k}}$$

Entonces:

$$\mathbf{F} = \mathbf{I}^{(\mathbf{n})} + \mathbf{A} \cdot h + \frac{(\mathbf{A} \cdot h)^2}{2}$$

Dominios de la Estabilidad Numérica de Algoritmos RK II

Los algoritmos tienen que aproximar la *solución analítica*:

$$\mathbf{F} = \exp(\mathbf{A} \cdot h) = \mathbf{I^{(n)}} + \mathbf{A} \cdot h + \frac{(\mathbf{A} \cdot h)^2}{2!} + \frac{(\mathbf{A} \cdot h)^3}{3!} + \ldots$$

hasta el término correspondiente al orden de la aproximación del método.

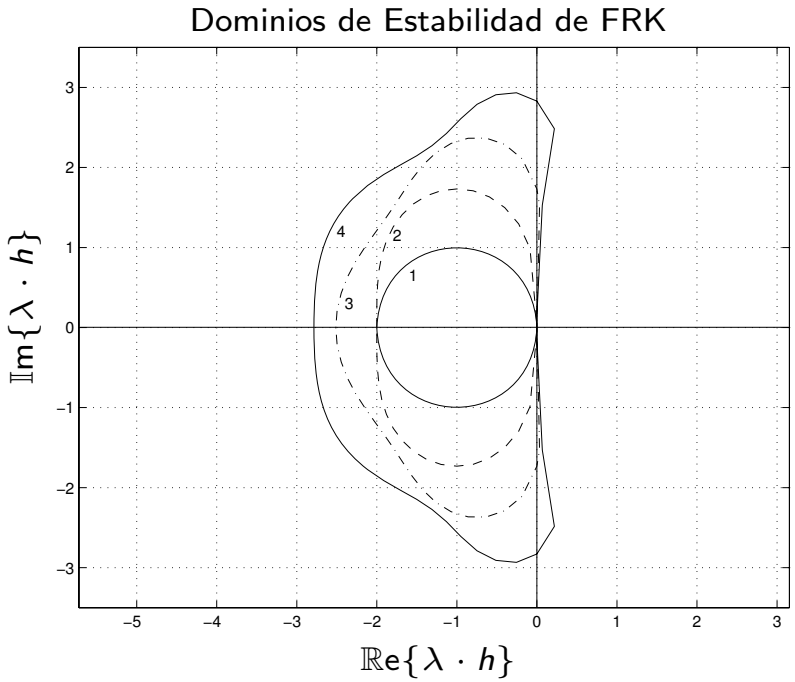

Figure: Dominios de estabilidad numérica de algoritmos RK explícitos

Sistemas Rígidos

- Llamamos a un sistema lineal *rígido* si es estable y tiene autovalores muy separados en términos de sus partes reales.

Sistemas Rígidos

- Llamamos a un sistema lineal *rígido* si es estable y tiene autovalores muy separados en términos de sus partes reales.
- Sistemas no lineales se llaman rígidos si son estables y tienen un comportamiento con modos rápidos y modos lentos. La linealización de esos sistemas produce sistemas lineales rígidos.

Sistemas Rígidos

- Llamamos a un sistema lineal *rígido* si es estable y tiene autovalores muy separados en términos de sus partes reales.
- Sistemas no lineales se llaman rígidos si son estables y tienen un comportamiento con modos rápidos y modos lentos. La linealización de esos sistemas produce sistemas lineales rígidos.
- Estos sistemas no pueden simularse eficientemente usando ningún algoritmo RK explícito porque se necesitarían pasos muy pequeños en el tiempo para llevar los autovalores (del sistema mismo o de su linealización) localizados muy a la izquierda del plano complejo $\boxed{\lambda \cdot h}$ dentro del dominio de la estabilidad numérica.

Sistemas Rígidos

- Llamamos a un sistema lineal *rígido* si es estable y tiene autovalores muy separados en términos de sus partes reales.
- Sistemas no lineales se llaman rígidos si son estables y tienen un comportamiento con modos rápidos y modos lentos. La linealización de esos sistemas produce sistemas lineales rígidos.
- Estos sistemas no pueden simularse eficientemente usando ningún algoritmo RK explícito porque se necesitarían pasos muy pequeños en el tiempo para llevar los autovalores (del sistema mismo o de su linealización) localizados muy a la izquierda del plano complejo $\boxed{\lambda \cdot h}$ dentro del dominio de la estabilidad numérica.
- De hecho es un problema con todos los algoritmos de integración numérica explícitos.

Sistemas Rígidos

- Llamamos a un sistema lineal *rígido* si es estable y tiene autovalores muy separados en términos de sus partes reales.
- Sistemas no lineales se llaman rígidos si son estables y tienen un comportamiento con modos rápidos y modos lentos. La linealización de esos sistemas produce sistemas lineales rígidos.
- Estos sistemas no pueden simularse eficientemente usando ningún algoritmo RK explícito porque se necesitarían pasos muy pequeños en el tiempo para llevar los autovalores (del sistema mismo o de su linealización) localizados muy a la izquierda del plano complejo $\boxed{\lambda \cdot h}$ dentro del dominio de la estabilidad numérica.
- De hecho es un problema con todos los algoritmos de integración numérica explícitos.
- **Para la simulación eficiente de sistemas rígidos se necesitan algoritmos implícitos.**

La Extrapolación de Richardson

Una manera para obtener algoritmos de órdenes más altos es combinar varias aproximaciones de órdenes más bajas.

Empezamos con un algoritmo FE, repetiendo el paso de t^* hasta $t^* + h$ cuatro veces con pasos de tamaños h, $h/2$, $h/3$ y $h/4$.

Aplicamos ese concepto al sistema lineal:

$$\begin{aligned}
\mathbf{x}^{\mathbf{P_1}}(k+1) &= [\mathbf{I}^{(\mathbf{n})} + \mathbf{A} \cdot h] \cdot \mathbf{x}(k) \\
\mathbf{x}^{\mathbf{P_2}}(k+1) &= [\mathbf{I}^{(\mathbf{n})} + \frac{\mathbf{A} \cdot h}{2}]^2 \cdot \mathbf{x}(k) \\
\mathbf{x}^{\mathbf{P_3}}(k+1) &= [\mathbf{I}^{(\mathbf{n})} + \frac{\mathbf{A} \cdot h}{3}]^3 \cdot \mathbf{x}(k) \\
\mathbf{x}^{\mathbf{P_4}}(k+1) &= [\mathbf{I}^{(\mathbf{n})} + \frac{\mathbf{A} \cdot h}{4}]^4 \cdot \mathbf{x}(k)
\end{aligned}$$

Usamos ahora la corrección parametrizada:

$$\mathbf{x}^{\mathbf{C}}(k+1) = \alpha_1 \cdot \mathbf{x}^{\mathbf{P_1}}(k+1) + \alpha_2 \cdot \mathbf{x}^{\mathbf{P_2}}(k+1) + \alpha_3 \cdot \mathbf{x}^{\mathbf{P_3}}(k+1) + \alpha_4 \cdot \mathbf{x}^{\mathbf{P_4}}(k+1)$$

La Extrapolación de Richardson II

Entonces obtenemos:

$$\begin{aligned}\mathbf{x^C}(k+1) =& [(\alpha_1+\alpha_2+\alpha_3+\alpha_4)\cdot \mathbf{I^{(n)}} \\ &+ (\alpha_1+\alpha_2+\alpha_3+\alpha_4)\cdot \mathbf{A}\cdot h \\ &+ (\frac{\alpha_2}{4}+\frac{\alpha_3}{3}+\frac{3\alpha_4}{8})\cdot(\mathbf{A}\cdot h)^2 \\ &+ (\frac{\alpha_3}{27}+\frac{\alpha_4}{16})\cdot(\mathbf{A}\cdot h)^3+\frac{\alpha_4}{256}\cdot(\mathbf{A}\cdot h)^4]\cdot \mathbf{x_k}\end{aligned}$$

Lo que queremos obtener es:

$$\mathbf{x^C}(k+1) = [\mathbf{I^{(n)}} + \mathbf{A}\cdot h + \frac{(\mathbf{A}\cdot h)^2}{2} + \frac{(\mathbf{A}\cdot h)^3}{6} + \frac{(\mathbf{A}\cdot h)^4}{24}]\cdot \mathbf{x_k}$$

Comparando los parámetros obtenemos cuatro ecuaciones en cuatro incógnitas:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1/4 & 1/3 & 3/8 \\ 0 & 0 & 1/27 & 1/16 \\ 0 & 0 & 0 & 1/256 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1/2 \\ 1/6 \\ 1/24 \end{pmatrix}$$

Entonces:

$$\alpha_1 = -\frac{1}{6} \;;\; \alpha_2 = 4 \;;\; \alpha_3 = -\frac{27}{2} \;;\; \alpha_4 = \frac{32}{3}$$

La Extrapolación de Richardson III

Encontramos otra implementación de un algoritmo RK4, pero

La Extrapolación de Richardson III

Encontramos otra implementación de un algoritmo RK4, pero

- No es cierto que ese algoritmo también es de cuarto orden para sistemas no lineales

La Extrapolación de Richardson III

Encontramos otra implementación de un algoritmo RK4, pero

- No es cierto que ese algoritmo también es de cuarto orden para sistemas no lineales
- No es una implementación eficiente porque usamos 10 etapas en lugar de cuatro.

La Extrapolación de Richardson IV

El orden del algoritmo no sería muy importante si pudiéramos usar pasos muy pequeños.

Podemos escribir:

$$\mathbf{x_{k+1}}(\eta) = \mathbf{x_{k+1}} + \mathbf{e_1} \cdot \eta + \mathbf{e_2} \cdot \frac{\eta^2}{2!} + \mathbf{e_3} \cdot \frac{\eta^3}{3!} + \dots$$

donde $\mathbf{x_{k+1}}$ es el *valor exacto pero desconocido* del vector de las variables de estado en el instante de tiempo $t^* + h$, mientras que $\mathbf{x_{k+1}}(\eta)$ es el *valor inexacto pero conocido* del mismo vector obtenido usando una integración con pasos de la longitud η. Desarrollamos ese valor inexacto en una serie de Taylor alrededor del valor exacto.

Hacemos cuatro experimentos diferentes con los mismos cuatro pasos de integración que antes:

$$\begin{aligned}
\mathbf{x^{P_1}}(\eta_1) &\approx \mathbf{x_{k+1}} + \mathbf{e_1} \cdot h + \frac{\mathbf{e_2}}{2!} \cdot h^2 + \frac{\mathbf{e_3}}{3!} \cdot h^3 \\
\mathbf{x^{P_2}}(\eta_2) &\approx \mathbf{x_{k+1}} + \mathbf{e_1} \cdot \frac{h}{2} + \frac{\mathbf{e_2}}{2!} \cdot (\frac{h}{2})^2 + \frac{\mathbf{e_3}}{3!} \cdot (\frac{h}{2})^3 \\
\mathbf{x^{P_3}}(\eta_3) &\approx \mathbf{x_{k+1}} + \mathbf{e_1} \cdot \frac{h}{3} + \frac{\mathbf{e_2}}{2!} \cdot (\frac{h}{3})^2 + \frac{\mathbf{e_3}}{3!} \cdot (\frac{h}{3})^3 \\
\mathbf{x^{P_4}}(\eta_4) &\approx \mathbf{x_{k+1}} + \mathbf{e_1} \cdot \frac{h}{4} + \frac{\mathbf{e_2}}{2!} \cdot (\frac{h}{4})^2 + \frac{\mathbf{e_3}}{3!} \cdot (\frac{h}{4})^3
\end{aligned}$$

La Extrapolación de Richardson V

Entonces:

$$\begin{pmatrix} \mathbf{x^{P_1}} \\ \mathbf{x^{P_2}} \\ \mathbf{x^{P_3}} \\ \mathbf{x^{P_4}} \end{pmatrix} \approx \begin{pmatrix} h^0 & h^1 & h^2 & h^3 \\ (h/2)^0 & (h/2)^1 & (h/2)^2 & (h/2)^3 \\ (h/3)^0 & (h/3)^1 & (h/3)^2 & (h/3)^3 \\ (h/4)^0 & (h/4)^1 & (h/4)^2 & (h/4)^3 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x_{k+1}} \\ \mathbf{e_1} \\ \mathbf{e_2}/2 \\ \mathbf{e_3}/6 \end{pmatrix}$$

Invertiendo la matriz de Vandermonde obtenemos:

$$\mathbf{x_{k+1}} \approx \begin{pmatrix} -\frac{1}{6} & 4 & -\frac{27}{2} & \frac{32}{3} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x^{P_1}} \\ \mathbf{x^{P_2}} \\ \mathbf{x^{P_3}} \\ \mathbf{x^{P_4}} \end{pmatrix}$$

Son los mismos parámetros que obtuvimos antes.

La Extrapolación de Richardson V

Entonces:

$$\begin{pmatrix} \mathbf{x^{P_1}} \\ \mathbf{x^{P_2}} \\ \mathbf{x^{P_3}} \\ \mathbf{x^{P_4}} \end{pmatrix} \approx \begin{pmatrix} h^0 & h^1 & h^2 & h^3 \\ (h/2)^0 & (h/2)^1 & (h/2)^2 & (h/2)^3 \\ (h/3)^0 & (h/3)^1 & (h/3)^2 & (h/3)^3 \\ (h/4)^0 & (h/4)^1 & (h/4)^2 & (h/4)^3 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x_{k+1}} \\ \mathbf{e_1} \\ \mathbf{e_2}/2 \\ \mathbf{e_3}/6 \end{pmatrix}$$

Invertiendo la matriz de Vandermonde obtenemos:

$$\mathbf{x_{k+1}} \approx \begin{pmatrix} -\frac{1}{6} & 4 & -\frac{27}{2} & \frac{32}{3} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x^{P_1}} \\ \mathbf{x^{P_2}} \\ \mathbf{x^{P_3}} \\ \mathbf{x^{P_4}} \end{pmatrix}$$

Son los mismos parámetros que obtuvimos antes.

Cada una de las dos derivaciones tiene sus ventajas y sus desventajas.

La Extrapolación de Richardson V

Entonces:

$$\begin{pmatrix} \mathbf{x^{P_1}} \\ \mathbf{x^{P_2}} \\ \mathbf{x^{P_3}} \\ \mathbf{x^{P_4}} \end{pmatrix} \approx \begin{pmatrix} h^0 & h^1 & h^2 & h^3 \\ (h/2)^0 & (h/2)^1 & (h/2)^2 & (h/2)^3 \\ (h/3)^0 & (h/3)^1 & (h/3)^2 & (h/3)^3 \\ (h/4)^0 & (h/4)^1 & (h/4)^2 & (h/4)^3 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x_{k+1}} \\ \mathbf{e_1} \\ \mathbf{e_2}/2 \\ \mathbf{e_3}/6 \end{pmatrix}$$

Invertiendo la matriz de Vandermonde obtenemos:

$$\mathbf{x_{k+1}} \approx \begin{pmatrix} -\frac{1}{6} & 4 & -\frac{27}{2} & \frac{32}{3} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x^{P_1}} \\ \mathbf{x^{P_2}} \\ \mathbf{x^{P_3}} \\ \mathbf{x^{P_4}} \end{pmatrix}$$

Son los mismos parámetros que obtuvimos antes.

Cada una de las dos derivaciones tiene sus ventajas y sus desventajas.

- El primer método garantizó que el método que resulta es de cuarto orden (al menos para sistemas lineales). El segundo método no nos produce ese resultado.

La Extrapolación de Richardson V

Entonces:

$$\begin{pmatrix} \mathbf{x^{P_1}} \\ \mathbf{x^{P_2}} \\ \mathbf{x^{P_3}} \\ \mathbf{x^{P_4}} \end{pmatrix} \approx \begin{pmatrix} h^0 & h^1 & h^2 & h^3 \\ (h/2)^0 & (h/2)^1 & (h/2)^2 & (h/2)^3 \\ (h/3)^0 & (h/3)^1 & (h/3)^2 & (h/3)^3 \\ (h/4)^0 & (h/4)^1 & (h/4)^2 & (h/4)^3 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x_{k+1}} \\ \mathbf{e_1} \\ \mathbf{e_2}/2 \\ \mathbf{e_3}/6 \end{pmatrix}$$

Invertiendo la matriz de Vandermonde obtenemos:

$$\mathbf{x_{k+1}} \approx \begin{pmatrix} -\frac{1}{6} & 4 & -\frac{27}{2} & \frac{32}{3} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x^{P_1}} \\ \mathbf{x^{P_2}} \\ \mathbf{x^{P_3}} \\ \mathbf{x^{P_4}} \end{pmatrix}$$

Son los mismos parámetros que obtuvimos antes.

Cada una de las dos derivaciones tiene sus ventajas y sus desventajas.

- El primer método garantizó que el método que resulta es de cuarto orden (al menos para sistemas lineales). El segundo método no nos produce ese resultado.
- Por otro lado, el primer método fue derivado usando explícitamente el algoritmo FE como método usado en los pasos internos. El segundo método no hace ninguna suposición. Los mismos parámetros resultarán usando cualquier método interno.

El Método IEX4

Usando la extrapolación de Richardson desarrollamos un *algoritmo implícito de cuarto orden*:

$$
\begin{array}{lll}
1^{\mathrm{ra}} \text{ predicción:} & \mathbf{k_1} & = \mathbf{x_k} + h \cdot \mathbf{f}(\mathbf{k_1}, t_{k+1}) \\
2^{\mathrm{da}} \text{ predicción:} & \mathbf{k_{2a}} & = \mathbf{x_k} + \frac{h}{2} \cdot \mathbf{f}(\mathbf{k_{2a}}, t_{k+\frac{1}{2}}) \\
 & \mathbf{k_2} & = \mathbf{k_{2a}} + \frac{h}{2} \cdot \mathbf{f}(\mathbf{k_2}, t_{k+1}) \\
3^{\mathrm{ra}} \text{ predicción:} & \mathbf{k_{3a}} & = \mathbf{x_k} + \frac{h}{3} \cdot \mathbf{f}(\mathbf{k_{3a}}, t_{k+\frac{1}{3}}) \\
 & \mathbf{k_{3b}} & = \mathbf{k_{3a}} + \frac{h}{3} \cdot \mathbf{f}(\mathbf{k_{3b}}, t_{k+\frac{2}{3}}) \\
 & \mathbf{k_3} & = \mathbf{k_{3b}} + \frac{h}{3} \cdot \mathbf{f}(\mathbf{k_3}, t_{k+1}) \\
4^{\mathrm{ta}} \text{ predicción:} & \mathbf{k_{4a}} & = \mathbf{x_k} + \frac{h}{4} \cdot \mathbf{f}(\mathbf{k_{4a}}, t_{k+\frac{1}{4}}) \\
 & \mathbf{k_{4b}} & = \mathbf{k_{4a}} + \frac{h}{4} \cdot \mathbf{f}(\mathbf{k_{4b}}, t_{k+\frac{1}{2}}) \\
 & \mathbf{k_{4c}} & = \mathbf{k_{4b}} + \frac{h}{4} \cdot \mathbf{f}(\mathbf{k_{4c}}, t_{k+\frac{3}{4}}) \\
 & \mathbf{k_4} & = \mathbf{k_{4c}} + \frac{h}{4} \cdot \mathbf{f}(\mathbf{k_4}, t_{k+1}) \\
\text{corrección:} & \mathbf{x_{k+1}} & = -\frac{1}{6} \cdot \mathbf{k_1} + 4 \cdot \mathbf{k_2} - \frac{27}{2} \cdot \mathbf{k_3} + \frac{32}{3} \cdot \mathbf{k_4}
\end{array}
$$

Lo llamamos IEX4.

El Método IEX4 II

Dibujamos el *dominio de la estabilidad numérica del algoritmo IEX4*:

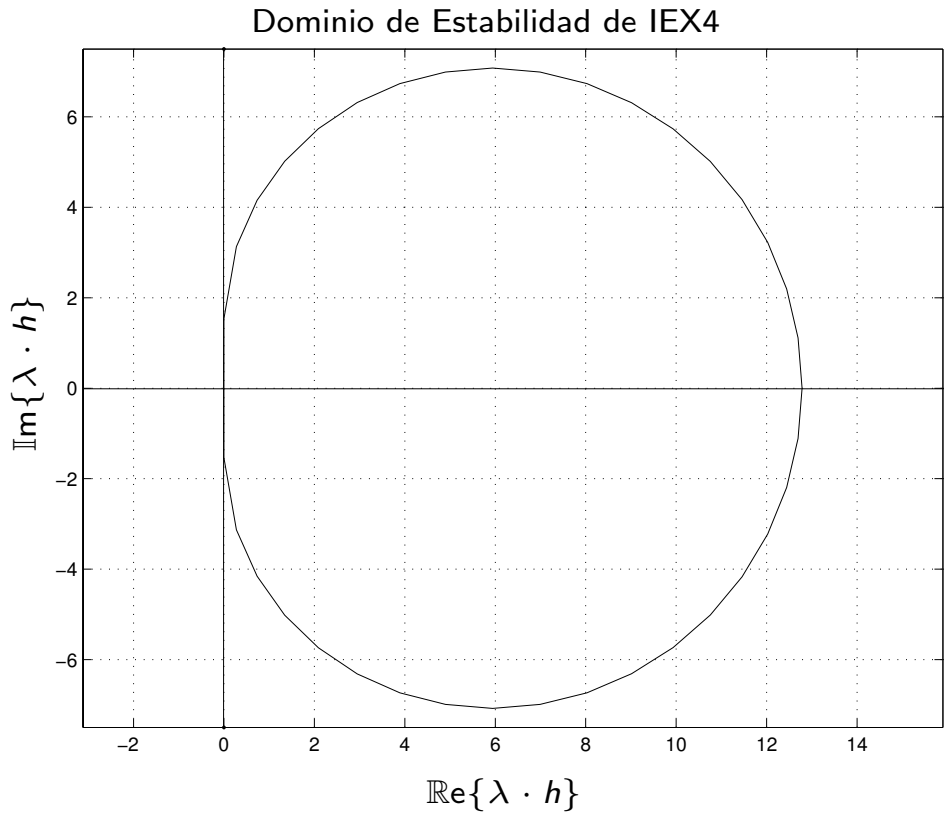

Figure: El dominio de la estabilidad numérica del algoritmo IEX4

Sistemas con Estabilidad Marginal

Si los *autovalores dominantes* de un sistema son complejos y se encuentran *cercano al eje imaginario*, tendremos problemas con la simulación.

Por un lado, los algoritmos explícitos necesitarán un paso muy pequeño para incluir esos autovalores en el dominio de la estabilidad numérica del algoritmo. Por otro lado, los algoritmos implícitos no nos producirán resultados precisos.

Lo que necesitaríamos son algoritmos que contienen en su dominio de la estabilidad numérica todo el plano complejo a la izquierda del eje imaginario y nada más.

Llamamos algoritmos de ese tipo F-estables (fielmente estables).

Los Métodos de Interpolación hacia Atrás

Miramos otra vez el algoritmo BE:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot \dot{\mathbf{x}}_{\mathbf{k+1}}$$

Al algoritmos BE puede reformularse de la forma siguiente:

$$\mathbf{x_k} = \mathbf{x_{k+1}} - h \cdot \dot{\mathbf{x}}_{\mathbf{k+1}}$$

Es decir, usar el algoritmo BE con paso h es igual a usar el algoritmo FE con paso $-h$.

Esa idea nos ofrece una nueva manera para implementar algoritmos implícitos. Empezamos con una estimación del valor $\mathbf{x_{k+1}}$ e integramos hacia atrás con un paso de $-h$, iterando sobre el valor $\mathbf{x_k}$ usando la iteración de Newton.

De esa manera podemos obtener *algoritmos implícitos de órdenes más altos* basados en los algoritmos FRK. **Llamamos esos algoritmos BRK.**

Los Algoritmos BRK

Los algoritmos BRK de órdenes 1..4 son caracterizados por:

$$\mathbf{F_1} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot h]^{-1}$$

$$\mathbf{F_2} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot h + \frac{(\mathbf{A} \cdot h)^2}{2!}]^{-1}$$

$$\mathbf{F_3} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot h + \frac{(\mathbf{A} \cdot h)^2}{2!} - \frac{(\mathbf{A} \cdot h)^3}{3!}]^{-1}$$

$$\mathbf{F_4} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot h + \frac{(\mathbf{A} \cdot h)^2}{2!} - \frac{(\mathbf{A} \cdot h)^3}{3!} + \frac{(\mathbf{A} \cdot h)^4}{4!}]^{-1}$$

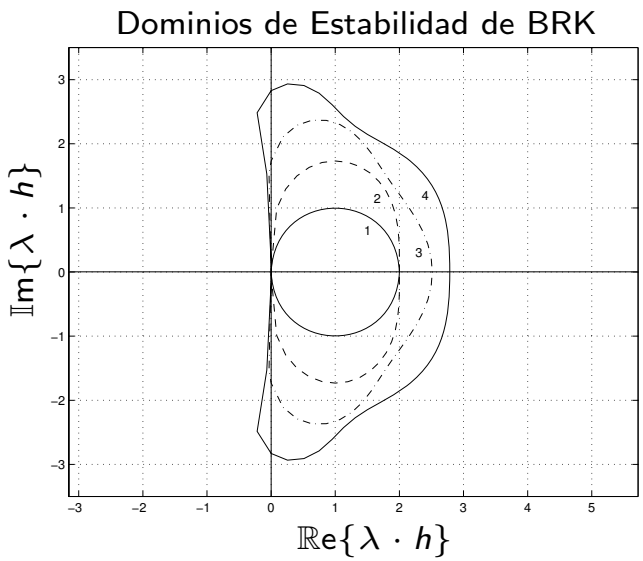

Figure: Dominios de la estabilidad numérica de los algoritmos BRK

Los Algoritmos BRK II

El algoritmo BRK4 es superior al algoritmo IEX4 porque:

Los Algoritmos BRK II

El algoritmo BRK4 es superior al algoritmo IEX4 porque:

- solamente necesita 4 etapas en lugar de 10,

Los Algoritmos BRK II

El algoritmo BRK4 es superior al algoritmo IEX4 porque:

- solamente necesita 4 etapas en lugar de 10,
- es garantizado de preservar su orden de aproximación también en el caso de sistemas no lineales y

Los Algoritmos BRK II

El algoritmo BRK4 es superior al algoritmo IEX4 porque:

- solamente necesita 4 etapas en lugar de 10,
- es garantizado de preservar su orden de aproximación también en el caso de sistemas no lineales y
- no necesita el desarrollo de software; se puede usar cualquiera implementación del algoritmo FRK4.

Los Algoritmos BI

La interpolación hacia atrás puede implementarse también de otra forma. Para ello desarrollamos un *método cíclico* que consiste en *medio paso FRK* (explícito) seguido por *medio paso BRK* (implícito). Iteramos sobre la diferencia entre los valores $\mathbf{x}_{\mathbf{k}+\frac{1}{2}}$:

$$\mathcal{F}(\mathbf{x}_{\mathbf{k}+1}) = \mathbf{x}^{\text{derecho}}_{\mathbf{k}+\frac{1}{2}} - \mathbf{x}^{\text{izquierdo}}_{\mathbf{k}+\frac{1}{2}} = 0.0$$

usando la iteración de Newton.

El algoritmo BI1 puede implementarse de la siguiente forma:

$$1^{\text{ra}} \text{ etapa:} \quad \mathbf{x}_{\mathbf{k}+\frac{1}{2}} = \mathbf{x}_{\mathbf{k}} + \frac{h}{2} \cdot \dot{\mathbf{x}}_{\mathbf{k}}$$

$$2^{\text{da}} \text{ etapa:} \quad \mathbf{x}_{\mathbf{k}+1} = \mathbf{x}_{\mathbf{k}+\frac{1}{2}} + \frac{h}{2} \cdot \dot{\mathbf{x}}_{\mathbf{k}+1}$$

Ese algoritmo también se conoce bajo el nombre *regla trapezoidal*. El método puede representarse por:

$$\mathbf{F}_{\mathbf{TR}} = [\mathbf{I}^{(\mathbf{n})} - \mathbf{A} \cdot \frac{h}{2}]^{-1} \cdot [\mathbf{I}^{(\mathbf{n})} + \mathbf{A} \cdot \frac{h}{2}]$$

Por casualidad, la *regla trapezoidal* es un *método de segundo orden*.

Los Algoritmos BI II

Los algoritmos BI pueden caracterizarse por:

$$\mathbf{F_{TR}} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot \frac{h}{2}]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A} \cdot \frac{h}{2}]$$

$$\mathbf{F_{BI2}} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot \frac{h}{2} + \frac{(\mathbf{A} \cdot h)^2}{8}]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A} \cdot \frac{h}{2} + \frac{(\mathbf{A} \cdot h)^2}{8}]$$

$$\mathbf{F_{BI3}} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot \frac{h}{2} + \frac{(\mathbf{A} \cdot h)^2}{8} - \frac{(\mathbf{A} \cdot h)^3}{48}]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A} \cdot \frac{h}{2} + \frac{(\mathbf{A} \cdot h)^2}{8} + \frac{(\mathbf{A} \cdot h)^3}{48}]$$

$$\mathbf{F_{BI4}} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot \frac{h}{2} + \frac{(\mathbf{A} \cdot h)^2}{8} - \frac{(\mathbf{A} \cdot h)^3}{48} + \frac{(\mathbf{A} \cdot h)^4}{384}]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A} \cdot \frac{h}{2} + \frac{(\mathbf{A} \cdot h)^2}{8} + \frac{(\mathbf{A} \cdot h)^3}{48} + \frac{(\mathbf{A} \cdot h)^4}{384}]$$

Todos los algoritmos BI son F-estables.

El algoritmo BI2 no es muy útil porque el método TR, que es más económico, ya es un algoritmo de segundo orden.

Las Aproximaciones de Padé

Padé trató aproximar numéricamente funciones exponenciales. Lo hizo de la forma siguiente:

$$\begin{aligned} \mathbf{F} &= \exp(\mathbf{A} \cdot h) \\ &= \exp(\mathbf{A}\frac{h}{2}) \cdot \exp(\mathbf{A}\frac{h}{2}) \\ &= [\exp(\mathbf{A}(-\frac{h}{2}))]^{-1} \cdot \exp(\mathbf{A}\frac{h}{2}) \end{aligned}$$

Encontró aproximaciones polinomiales para las dos expresiones exponenciales:

$$\mathbf{F} \approx \mathbf{D}(p,q)^{-1} \cdot \mathbf{N}(p,q)$$

donde:

$$\begin{aligned} \mathbf{D}(p,q) &= \sum_{j=0}^{q} \frac{(p+q-j)!\ q!}{(p+q)!\ j!\ (q-j)!} \cdot (-\mathbf{A}h)^j \\ \mathbf{N}(p,q) &= \sum_{j=0}^{p} \frac{(p+q-j)!\ p!}{(p+q)!\ j!\ (p-j)!} \cdot (\mathbf{A}h)^j \end{aligned}$$

Las Aproximaciones de Padé II

Para $p = q$ obtenemos los algoritmos:

$$\mathbf{F_{TR}} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot \frac{h}{2}]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A} \cdot \frac{h}{2}]$$

$$\mathbf{F_{P4}} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot \frac{h}{2} + \frac{(\mathbf{A} \cdot h)^2}{12}]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A} \cdot \frac{h}{2} + \frac{(\mathbf{A} \cdot h)^2}{12}]$$

$$\mathbf{F_{P6}} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot \frac{h}{2} + \frac{(\mathbf{A} \cdot h)^2}{10} - \frac{(\mathbf{A} \cdot h)^3}{120}]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A} \cdot \frac{h}{2} + \frac{(\mathbf{A} \cdot h)^2}{10} + \frac{(\mathbf{A} \cdot h)^3}{120}]$$

$$\mathbf{F_{P8}} = [\mathbf{I^{(n)}} - \mathbf{A} \cdot \frac{h}{2} + \frac{3(\mathbf{A} \cdot h)^2}{28} - \frac{(\mathbf{A} \cdot h)^3}{84} + \frac{(\mathbf{A} \cdot h)^4}{1680}]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A} \cdot \frac{h}{2} + \frac{3(\mathbf{A} \cdot h)^2}{28} + \frac{(\mathbf{A} \cdot h)^3}{84} + \frac{(\mathbf{A} \cdot h)^4}{1680}]$$

Los *algoritmos de Padé* son muy similares a los *algoritmos BI*. Todos estos algoritmos son también F-estables. Por la *misma carga computacional* es posible *doblar el orden de aproximación* de ese tipo de algoritmos.

Las Aproximaciones de Padé III

Se pudiera pensar que los algoritmos Padé son superiores a los algoritmos BI a causa del orden más alto de la aproximación que aportan con la misma carga computacional.

Desafortunadamente no es el caso. Mientras que el algoritmo BI4 es de cuarto orden para todos los sistemas, el algoritmo P8 es de octavo orden para sistemas lineales y de segundo orden para sistemas no lineales.

Por eso, los algoritmos Padé no deben usarse salvo para la simulación de sistemas lineales.

Algoritmos para la Simulación de Sistemas Rígidos

Un autovalor:

$$\lambda = \sigma + j \cdot \omega$$

produce una trayectoria con un *factor de amortiguación* de $-\sigma$ y una *frecuencia de oscilación* de ω.

Por eso, las trayectorias resultantes de autovalores localizados muy a la izquierda del eje imaginario del plano complejo $\boxed{\lambda \cdot h}$ deben amortiguarse rápidamente.

Algoritmos para la Simulación de Sistemas Rígidos

Un autovalor:

$$\lambda = \sigma + j \cdot \omega$$

produce una trayectoria con un *factor de amortiguación* de $-\sigma$ y una *frecuencia de oscilación* de ω.

Por eso, las trayectorias resultantes de autovalores localizados muy a la izquierda del eje imaginario del plano complejo $\boxed{\lambda \cdot h}$ deben amortiguarse rápidamente.

- Desafortunadamente, los *algoritmos F-estables* no producen ese comportamiento. Esos algoritmos tienen estabilidad marginal en todo el eje imaginario hasta el infinito. Entonces tienen estabilidad marginal en el infinito independientemente de la dirección de acercamiento.

Algoritmos para la Simulación de Sistemas Rígidos

Un autovalor:

$$\lambda = \sigma + j \cdot \omega$$

produce una trayectoria con un *factor de amortiguación* de $-\sigma$ y una *frecuencia de oscilación* de ω.

Por eso, las trayectorias resultantes de autovalores localizados muy a la izquierda del eje imaginario del plano complejo $\boxed{\lambda \cdot h}$ deben amortiguarse rápidamente.

- Desafortunadamente, los *algoritmos F-estables* no producen ese comportamiento. Esos algoritmos tienen estabilidad marginal en todo el eje imaginario hasta el infinito. Entonces tienen estabilidad marginal en el infinito independientemente de la dirección de acercamiento.
- En lugar de ofrecer una amortiguación muy grande de esos autovalores, casi no tienen ninguna.

Algoritmos para la Simulación de Sistemas Rígidos

Un autovalor:

$$\lambda = \sigma + j \cdot \omega$$

produce una trayectoria con un *factor de amortiguación* de $-\sigma$ y una *frecuencia de oscilación* de ω.

Por eso, las trayectorias resultantes de autovalores localizados muy a la izquierda del eje imaginario del plano complejo $\boxed{\lambda \cdot h}$ deben amortiguarse rápidamente.

- Desafortunadamente, los *algoritmos F-estables* no producen ese comportamiento. Esos algoritmos tienen estabilidad marginal en todo el eje imaginario hasta el infinito. Entonces tienen estabilidad marginal en el infinito independientemente de la dirección de acercamiento.
- En lugar de ofrecer una amortiguación muy grande de esos autovalores, casi no tienen ninguna.
- **Por esa razón, los algoritmos F-estables no sirven para la simulación de sistemas rígidos.**

Propiedades de la Estabilidad Numérica

Los algoritmos que funcionan bien para la simulación de sistemas rígidos deben tener un dominio de la estabilidad numérica similar al dominio del algoritmo BE, es decir cerrado en la parte derecha del plano complejo $\boxed{\lambda \cdot h}$.

Propiedades de la Estabilidad Numérica

Los algoritmos que funcionan bien para la simulación de sistemas rígidos deben tener un dominio de la estabilidad numérica similar al dominio del algoritmo BE, es decir cerrado en la parte derecha del plano complejo $\lambda \cdot h$.

- **Un algoritmo que contiene toda la parte izquierda del plano complejo $\lambda \cdot h$ dentro de su dominio de estabilidad numérica se llama A-estable (absolutamente estable).**

Propiedades de la Estabilidad Numérica

Los algoritmos que funcionan bien para la simulación de sistemas rígidos deben tener un dominio de la estabilidad numérica similar al dominio del algoritmo BE, es decir cerrado en la parte derecha del plano complejo $\lambda \cdot h$.

- **Un algoritmo que contiene toda la parte izquierda del plano complejo $\lambda \cdot h$ dentro de su dominio de estabilidad numérica se llama A-estable (absolutamente estable).**
- Los algoritmos que funcionan bien para la simulación de sistemas rígidos deben ser A-estables. Sin embargo, esa propiedad solamente es necesaria, pero no es suficiente.

Propiedades de la Estabilidad Numérica

Los algoritmos que funcionan bien para la simulación de sistemas rígidos deben tener un dominio de la estabilidad numérica similar al dominio del algoritmo BE, es decir cerrado en la parte derecha del plano complejo $\lambda \cdot h$.

- **Un algoritmo que contiene toda la parte izquierda del plano complejo $\lambda \cdot h$ dentro de su dominio de estabilidad numérica se llama A-estable (absolutamente estable).**
- Los algoritmos que funcionan bien para la simulación de sistemas rígidos deben ser A-estables. Sin embargo, esa propiedad solamente es necesaria, pero no es suficiente.
- **Un algoritmo que es A-estable y además ofrece un factor de amortiguación infinito en el infinito se llama L-estable.**

Propiedades de la Estabilidad Numérica

Los algoritmos que funcionan bien para la simulación de sistemas rígidos deben tener un dominio de la estabilidad numérica similar al dominio del algoritmo BE, es decir cerrado en la parte derecha del plano complejo $\lambda \cdot h$.

- **Un algoritmo que contiene toda la parte izquierda del plano complejo $\lambda \cdot h$ dentro de su dominio de estabilidad numérica se llama A-estable (absolutamente estable).**
- Los algoritmos que funcionan bien para la simulación de sistemas rígidos deben ser A-estables. Sin embargo, esa propiedad solamente es necesaria, pero no es suficiente.
- **Un algoritmo que es A-estable y además ofrece un factor de amortiguación infinito en el infinito se llama L-estable.**
- Los algoritmos F-estables son también A-estables, pero no son L-estables.

Propiedades de la Estabilidad Numérica

Los algoritmos que funcionan bien para la simulación de sistemas rígidos deben tener un dominio de la estabilidad numérica similar al dominio del algoritmo BE, es decir cerrado en la parte derecha del plano complejo $\lambda \cdot h$.

- **Un algoritmo que contiene toda la parte izquierda del plano complejo $\lambda \cdot h$ dentro de su dominio de estabilidad numérica se llama A-estable (absolutamente estable).**
- Los algoritmos que funcionan bien para la simulación de sistemas rígidos deben ser A-estables. Sin embargo, esa propiedad solamente es necesaria, pero no es suficiente.
- **Un algoritmo que es A-estable y además ofrece un factor de amortiguación infinito en el infinito se llama L-estable.**
- Los algoritmos F-estables son también A-estables, pero no son L-estables.
- **Se desean algoritmos L-estables para la simulación de sistemas rígidos.**

Algoritmos BI para la Simulación de Sistemas Rígidos

Se pueden generar *algoritmos del tipo BI* con dominios de la estabilidad numérica similares al dominio de BE de la manera siguiente.

En lugar de efectuar el paso FRK hasta el punto medio del intervalo seguido por un paso BRK del mismo tamaño, es posible avanzar un paso FRK de longitud $\vartheta \cdot h$ con $\vartheta < 0.5$ seguido por un paso BRK de la longitud $(1-\vartheta) \cdot h$:

$$\mathbf{F_{BI1}}(\vartheta) = [\mathbf{I^{(n)}} - \mathbf{A}(1-\vartheta)h]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A}\vartheta h]$$

$$\mathbf{F_{BI2}}(\vartheta) = [\mathbf{I^{(n)}} - \mathbf{A}(1-\vartheta)h + \frac{(\mathbf{A}(1-\vartheta)h)^2}{2!}]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A}\vartheta h + \frac{(\mathbf{A}\vartheta h)^2}{2!}]$$

$$\mathbf{F_{BI3}}(\vartheta) = [\mathbf{I^{(n)}} - \mathbf{A}(1-\vartheta)h + \frac{(\mathbf{A}(1-\vartheta)h)^2}{2!} - \frac{(\mathbf{A}(1-\vartheta)h)^3}{3!}]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A}\vartheta h + \frac{(\mathbf{A}\vartheta h)^2}{2!} + \frac{(\mathbf{A}\vartheta h)^3}{3!}]$$

$$\mathbf{F_{BI4}}(\vartheta) = [\mathbf{I^{(n)}} - \mathbf{A}(1-\vartheta)h + \frac{(\mathbf{A}(1-\vartheta)h)^2}{2!} - \frac{(\mathbf{A}(1-\vartheta)h)^3}{3!} + \frac{(\mathbf{A}(1-\vartheta)h)^4}{4!}]^{-1} \cdot [\mathbf{I^{(n)}} + \mathbf{A}\vartheta h + \frac{(\mathbf{A}\vartheta h)^2}{2!} + \frac{(\mathbf{A}\vartheta h)^3}{3!} + \frac{(\mathbf{A}\vartheta h)^4}{4!}]$$

Algoritmos BI para la Simulación de Sistemas Rígidos II

Con:

$$\vartheta = 0.4$$

obtenemos los algoritmos:

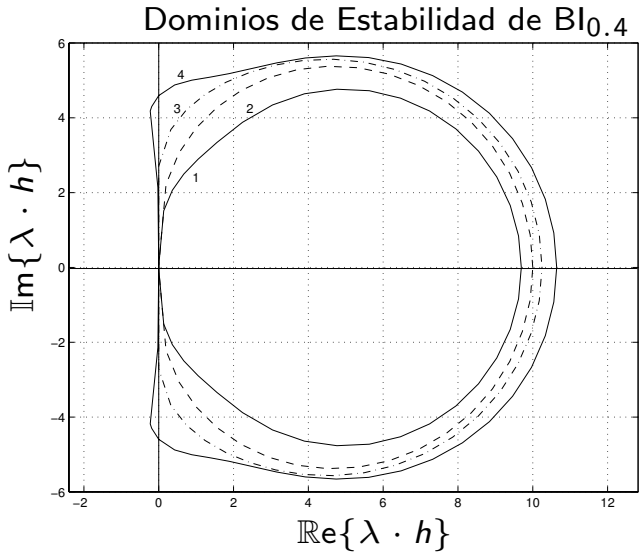

Figure: Dominios de la estabilidad numérica de los algoritmos $BI_{0.4}$

Todos estos algoritmos con la excepción de $BI4_{0.4}$ son A-estables, pero ninguno entre ellos es L-estable.

Algoritmos BI para la Simulación de Sistemas Rígidos III

Para obtener algoritmos L-estables es necesario que el polinomio del denominador dea de un orden más alto que el polinomio del numerador.

Una manera de obtener algoritmos del tipo BI que son L-estables es usar un algoritmo de un orden más alto para su medio paso BRK.

Un muy buen algoritmo de ese tipo es el algoritmo $BI4/5_{0.45}$. Ese algoritmo empieza con medio paso FRK4 de longitud $0.45 \cdot h$ seguido por medio paso BRK5 de longitud $0.55 \cdot h$.

Conclusiones

Conclusiones

- En esta presentación introdujimos los *algoritmos Runge Kutta* que son algoritmos en un solo paso con múltiples evaluaciones de las ecuaciones del estado dentro del paso.

Conclusiones

- En esta presentación introdujimos los *algoritmos Runge Kutta* que son algoritmos en un solo paso con múltiples evaluaciones de las ecuaciones del estado dentro del paso.
- Existen algoritmos RK que son *explícitos* y otros que son *implícitos*.

Conclusiones

- En esta presentación introdujimos los *algoritmos Runge Kutta* que son algoritmos en un solo paso con múltiples evaluaciones de las ecuaciones del estado dentro del paso.
- Existen algoritmos RK que son *explícitos* y otros que son *implícitos*.
- Hablamos de las *propiedades de la estabilidad numérica* de algoritmos para la simulación, como la *F-estabilidad*, la *A-estabilidad* y la *L-estabilidad*.

Conclusiones

- En esta presentación introdujimos los *algoritmos Runge Kutta* que son algoritmos en un solo paso con múltiples evaluaciones de las ecuaciones del estado dentro del paso.
- Existen algoritmos RK que son *explícitos* y otros que son *implícitos*.
- Hablamos de las *propiedades de la estabilidad numérica* de algoritmos para la simulación, como la *F-estabilidad*, la *A-estabilidad* y la *L-estabilidad*.
- Introdujimos unos *algoritmos combinados* que consisten de múltiples medio pasos FRK y/o BRK como los algoritmos de la *extrapolación de Richardson*.

Conclusiones

- En esta presentación introdujimos los *algoritmos Runge Kutta* que son algoritmos en un solo paso con múltiples evaluaciones de las ecuaciones del estado dentro del paso.
- Existen algoritmos RK que son *explícitos* y otros que son *implícitos*.
- Hablamos de las *propiedades de la estabilidad numérica* de algoritmos para la simulación, como la *F-estabilidad*, la *A-estabilidad* y la *L-estabilidad*.
- Introdujimos unos *algoritmos combinados* que consisten de múltiples medio pasos FRK y/o BRK como los algoritmos de la *extrapolación de Richardson*.
- Introdujimos además unos *algoritmos cíclicos* que consisten de múltiples medio pasos FRK y/o BRK como los algoritmos de la *interpolación hacia atrás*.