Compito 4b: metodi RK impliciti

2020-2021 APPROSSIMAZIONE NUMERICA DI ODEs

Primo Passo: Implementare in MATLAB la seguenti funzioni:

1. `function [y,nevals,iter,nfail]=RK_implicit(fun,jac,t0,tf,y0,tableau,maxiter,TOL)`
2. `function [y,nevals,iter,nfail]=Gauss(s,fun,jac,t0,tf,y0,,maxiter,TOL)`
3. `function [y,nevals,iter,nfail]=LobattoA(s,fun,jac,t0,tf,y0,maxiter,TOL)`
4. `function [y,nevals,iter,nfail]=LobattoB(s,fun,jac,t0,tf,y0, maxiter,TOL)`
5. `function [y,nevals,iter,nfail]=LobattoC(s,fun,jac,t0,tf,y0, maxiter,TOL)`
6. `function [y,nevals,iter,nfail]=RadauA(s,fun,jac,t0,tf,y0,maxiter,TOL)`

corrispondenti ai metodi RUNGE-KUTTA impliciti per approssimare il problema di valori iniziali:

$$\begin{cases} y' = f(t,y) \\ y(t_0) = y_0 \end{cases}$$

dove $y \in \mathbb{R}^d$ and $f : [0, t_f] \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$ con $d \geq 1$ (il codice dovrebbe funzionare per problemi scalari e sistemi!!).

La evoluzione temporale richiede ad ogni paso di tempo, la risoluzione di un sistema di equazioni non-lineari tramite il metodo di Newton.

I diversi metodi RK impliciti sono definiti via:

$$g_i = y_n + h \sum_{j=1}^{s} a_{ij}\, f(t_n + h_n\, c_j\, ,\, g_j), \tag{0.1}$$

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i\, f(t_n + h_n\, c_i\, ,\, g_i). \tag{0.2}$$

Per la implementazione del metodo, e consigliato introdurre in (0.1) e (0.2) le quantità:

$$z_i = g_i - y_n \qquad i = 1, \dots s.$$

In questo modo, le equazioni per il metodo possono essere scritte come:

$$\begin{aligned} z_i &= \sum_{j=1}^{s} a_{ij}\ h \cdot\ f(t_n + h_n\, c_j\ ,\ y_n + z_j), \\ y_{n+1} &= y_n + \sum_{i=1}^{s} b_i\ h \cdot\ f(t_n + h_n\, c_i\ ,\ y_n + z_i). \end{aligned} \tag{0.3}$$

Definendo ora

$$\mathbf{z} = [z_1, \dots z_s]^T \qquad \mathbf{F}(\mathbf{z}) := \begin{bmatrix} hf(t_n + h\, c_1\ ,\ y_n + z_1) \\ hf(t_n + h\, c_2\ ,\ y_n + z_2) \\ \vdots \\ hf(t_n + h\, c_s\ ,\ y_n + z_s) \end{bmatrix} \qquad \mathbf{z} = [z_1, \dots z_s]^T \tag{0.4}$$

Le equazioni in (0.3) si possono scrivere in forma matriciale come:

$$\mathbf{z} = A\mathbf{F}(\mathbf{z}), \qquad \Longrightarrow \qquad y_{n+1} = y_n + b^T \left(A^{-1}\mathbf{z}\right)\ , \tag{0.5}$$

dove A e b sono specificati nel tableau di Butcher che definisce il metodo. Osservare che $\mathbf{z}$ e definita implicitamente tramite la prima equazione (e quindi si deve risolvere prima un sistema -possibilmente non lineare-). Notare inoltre che con questa scrittura si evitano s valutazioni di funzione al calcolare y_{n+1}.
Per risolvere il sistema in (0.5) si suggerisce usare iterazione di Newton (o Newton modificato).

Si suggerisce implementare la forma (0.5) (corrispondente (0.3)) con lo scopo di ridurre la possibile influenza degli errori di arrotondamento (round-off errors) e dovuti alla iterazione di Newton. Inoltre, la ulteriore valutazione di $\mathbf{z}$ in (0.2) coinvolgerebbe una eventuale amplificazione di detti errori; e nel caso che f abbia una costante di Lipschitz grande (problemi Stiff, per esempio) implicherebbe una grande amplificazioni di detti errori che potrebbe risultare disastrosa.

Sintassi. Variabili di Input e output

Variabili di Input

`fun` Variable di caratteri e una "function handle".
La funzione *[y_b]=fun(t_a, y_a)* valuta la funzione f (lato destro della ODE) nel instante di tempo t_a e valore y_a (cioe valuta la funzione f nel punto (t_a, y_a) del piano di fasi). Osservare che per un sistema il valore in output deve essere sempre vettoriale. Per ogni problema, si deve costruire un *.m* -file diverso che contiene la funzione specifica del problema.

`jac` Variable di caratteri e una "function handle".
La funzione *[G_b]=jac(t_a, y_a)* valuta il Jacobiano della funzione f (Differenziale con rispetto a y del lato destro della ODE) nel instante di tempo t_a e valore y_a (cioe valuta la funzione $D_y f$ nel punto (t_a, y_a) del piano di fasi). Osservare che per un sistema il valore in output deve essere sempre una matrice. Per ogni problema, si deve costruire un *.m* -file diverso che contiene il Jacobiano della funzione che specifica del problema.

$t0$ tempo iniziale

tf tempo finale

h grandezza del paso di tempo con cui si vuole avanzare nella evoluzione.

`TOL` Tolleranza per la iterazione di Newton

maxiter numero massimo di iterazioni permesse per la iterazione di Newton

y_0 vettore colonna che contiene il dato iniziale y_0

`tableau` Variable di caratteri (e una "function handle") che contiene il nome del file `.m`-dove vengono definiti i coefficienti del tableau di BUTCHER:

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}$$

corrispondente al metodo RK implicito che si vuole usare nell'approssimazione.

Output variables

`y`] column vector containing the approximate solution at time t_f

`nevals` Costo del algoritmo in termini di numero di valutazioni di funzione.

`nstep` numero di passi

`nfail` numero di fallimenti della iterazione di Newton

Risoluzione dei sistemi nonlineari

Ad ogni paso di tempo, i metodi IRK (RK implicito) richiedono la risoluzione del sistema (0.5) che eventualmente sará non lineare (se f e non lineare). La risoluzione di detto sistema si fa tramite un metodo iterativo. Si e osservato che una iterazione di punto fisso per (0.5) renderebbe il avanzamento in tempo esplicito. Per questa ragione si consiglia di usare iterazione di Newton. La iterazione di Newton per (0.5), agisce su $I - A\mathbf{F}(\mathbf{z}) \approx 0$ e si scrive

$$\left(I - A\frac{\partial F}{\partial y}(\mathbf{z}^{(k)})\right)\left(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}\right) = -\mathbf{z}^{(k)} + A\mathbf{F}(\mathbf{z}^{(\mathbf{k})}), \qquad k = 0, 1, \ldots \tag{0.6}$$

La matrice del sistema (lineare) risulta

(0.7)

$$\left(I - A\frac{\partial F}{\partial y}(\mathbf{z}^{(k)})\right) = \begin{pmatrix} I - ha_{11}\frac{\partial f}{\partial y}(t_0 + c_1 h, y_0 + z_1^{(k)}) & \ldots & -ha_{1s}\frac{\partial f}{\partial y}(t_0 + c_s h, y_0 + z_s^{(k)}) \\ \vdots & \ddots & \vdots \\ -ha_{s1}\frac{\partial f}{\partial y}(t_0 + c_1 h, y_0 + z_1^{(k)}) & \ldots & I - ha_{ss}\frac{\partial f}{\partial y}(t_0 + c_s h, y_0 + z_s^{(k)}) \end{pmatrix}$$

Osservare che la matrice sopra ha dimensioni $\mathbb{R}^{ds\times ds}$ dove s é il numero di tappe del IRK e d la dimensione del problema che vogliamo risolvere; la matrice

$$\frac{\partial f}{\partial y}(t_0 + c_i h, y_0 + z_i^{(k)}) \in \mathbb{R}^{d\times d}\ , \quad i = 1, \ldots s\ , \tag{0.8}$$

rappresenta la matrice Jacobiana di f valutata nel punto $(t_0 + c_1 h, y_0 + z_1^{(k)})$. Tipicamente, nelle applicazioni, al posto di ri-calcolare ad ogni passo di tempo, tutte le matrici Jacobiana per la iterazione di Newton e molto piu vantaggioso (in termini di costo) usare Newton modificato. Cioè, in (0.7) si rimpiazzano le matrici Jacobiane in (0.8) per la corrispondente matrici Jacobiana valutata nel punto iniziale (t_0, y_0) (e di conseguenza per $\mathbf{z}^{(0)} = 0$) e usare detta matrice per (la iterazione di Newton coinvolta in) ulteriori passi in tempo fino a che la iterazione di Newton fallisca. Solo quando Newton fallisce si deve ri-calcolare il Jacobiano (ora nel punto (t_n, y_n) dove ha fallito Newton) e aggiornare con i nuovi Jacobiani la matrice in (0.7). Quindi, con la definizione

$$J_0 := \frac{\partial f}{\partial y}(t_0, y_0), \tag{0.9}$$

la iterazione di Newton modificato risulta finalmente

$$(I - hA \otimes J_0)\left(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}\right) = -\mathbf{z}^{(k)} + (A \otimes I)\mathbf{F}(\mathbf{z}^{(\mathbf{k})}), \qquad k = 0, 1, \ldots \tag{0.10}$$

Per la implementazione del sistema (0.10) sono utili le funzioni *kron* e *reshape* di MATLAB.

Come iterante iniziale per la iterazione, basta prendere $\mathbf{z}^{(0)} = 0$. Altre scelte piú sofisticate sono possibili, vedere Hairer II pp. 121. La tolleranza per Newton si fissa in funzione della accuratezza che si desidera per risolvere la integrazione in tempo. Indicativamente sempre maggiore.

La risoluzione del sistema lineare in (0.10) si púo fare se la dimensione del problema lo permette usando la fattorizzazione LU. In generale detta fattorizzazione deve essere calcolata solo una volta (o tante volte come fallimenti di Newton ci siano) e di solito viene pre-calcolata. In modo che la parte piu costosa si fa il menor numero di volte possibili.

In caso che il problema sia di grande dimensioni (per esempio se proviene della discretizzazione spaziale di un problema di PDEs) puo risultare piu vantaggioso (e a volte necessario) sostituire il solutore diretto con un solutore iterativo tipo CG (Gradiente Conjugato) o GMREs (Generalized Minimum residual) possibilmente precondizionato.