Soluzioni Analitiche e Numeriche Applicate all'Ingegneria Ambientale

Massimiliano Martinelli
massimiliano.martinelli@gmail.com

Università Politecnica delle Marche, Ancona
Facoltà di Ingegneria

4-5 Febbraio 2009

Outline

Outline

Outline

Sistemi non-lineari

Il problema

Data una funzione $\mathbf{F}\colon D \subseteq \mathbb{R}^n \to \mathbb{R}^n$, trovare $\mathbf{x}^* \in \mathbb{R}^n$ tale che

$$\mathbf{F}(\mathbf{x}^*) = \mathbf{0} \tag{1}$$

Sistemi non-lineari

Il problema

Data una funzione $\mathbf{F}\colon D \subseteq \mathbb{R}^n \to \mathbb{R}^n$, trovare $\mathbf{x}^* \in \mathbb{R}^n$ tale che

$$\mathbf{F}(\mathbf{x}^*) = \mathbf{0} \tag{1}$$

Motivazione

Discretizzazione delle equazioni alle derivate parziali non lineari $\Rightarrow$ sistemi di equazioni non lineari

Sistemi non-lineari

Il problema

Data una funzione $\mathbf{F}\colon D \subseteq \mathbb{R}^n \to \mathbb{R}^n$, trovare $\mathbf{x}^* \in \mathbb{R}^n$ tale che

$$\mathbf{F}(\mathbf{x}^*) = \mathbf{0} \tag{1}$$

Motivazione

Discretizzazione delle equazioni alle derivate parziali non lineari $\Rightarrow$ sistemi di equazioni non lineari

- Il problema seguente:

$$\text{trovare } \mathbf{x}^* \in \mathbb{R}^n \text{ tale che} \quad \widehat{\mathbf{F}}(\mathbf{x}^*) = \mathbf{y} \quad \text{per } \mathbf{y} \in \mathbb{R}^n$$

può essere scritto nella forma (1) semplicemente ponendo $\mathbf{F}(\mathbf{x}) = \widehat{\mathbf{F}}(\mathbf{x}) - \mathbf{y}$

Sistemi non-lineari

Il problema

Data una funzione $\mathbf{F}\colon D \subseteq \mathbb{R}^n \to \mathbb{R}^n$, trovare $\mathbf{x}^* \in \mathbb{R}^n$ tale che

$$\mathbf{F}(\mathbf{x}^*) = \mathbf{0} \tag{1}$$

Motivazione

Discretizzazione delle equazioni alle derivate parziali non lineari $\Rightarrow$ sistemi di equazioni non lineari

- Il problema seguente:

  $$\text{trovare } \mathbf{x}^* \in \mathbb{R}^n \text{ tale che} \quad \widehat{\mathbf{F}}(\mathbf{x}^*) = \mathbf{y} \quad \text{per } \mathbf{y} \in \mathbb{R}^n$$

  può essere scritto nella forma (1) semplicemente ponendo $\mathbf{F}(\mathbf{x}) = \widehat{\mathbf{F}}(\mathbf{x}) - \mathbf{y}$
- Il problema della soluzione di un sistema di equazioni non lineari può essere sempre riformulato come un problema di minimizzazione

Sistemi non-lineari

- Al contrario del caso lineare non c'è una teoria generale che assicura l'esistenza e l'unicità della soluzione

Sistemi non-lineari

- Al contrario del caso lineare non c'è una teoria generale che assicura l'esistenza e l'unicità della soluzione
- La risoluzione numerica è sempre basata su un algoritmo iterativo

Sistemi non-lineari

- Al contrario del caso lineare non c'è una teoria generale che assicura l'esistenza e l'unicità della soluzione
- La risoluzione numerica è sempre basata su un algoritmo iterativo
- In generale, dato $\mathbf{x}^{(0)}$, la procedura iterativa consiste nel definire una successione $\mathbf{x}^{(k)}$

$$\mathbf{x}^{(k+1)} = \mathbf{G}(\mathbf{x}^{(k)}) \qquad k \geq 0$$

in cui l'operatore $\mathbf{G}\colon \mathbb{R}^n \to \mathbb{R}^n$ verifica

$$\mathbf{G}(\mathbf{x}^*) = \mathbf{x}^* \quad \text{ogni volta che} \quad \mathbf{F}(\mathbf{x}^*) = \mathbf{0}$$

Sistemi non-lineari

- Al contrario del caso lineare non c'è una teoria generale che assicura l'esistenza e l'unicità della soluzione
- La risoluzione numerica è sempre basata su un algoritmo iterativo
- In generale, dato $\mathbf{x}^{(0)}$, la procedura iterativa consiste nel definire una successione $\mathbf{x}^{(k)}$

$$\mathbf{x}^{(k+1)} = \mathbf{G}(\mathbf{x}^{(k)}) \qquad k \geq 0$$

in cui l'operatore $\mathbf{G}\colon \mathbb{R}^n \to \mathbb{R}^n$ verifica

$$\mathbf{G}(\mathbf{x}^*) = \mathbf{x}^* \quad \text{ogni volta che} \quad \mathbf{F}(\mathbf{x}^*) = \mathbf{0}$$

Esempio

$$\mathbf{G}(\mathbf{x}) = \mathbf{x} + A(\mathbf{x})\mathbf{F}(\mathbf{x})$$

in cui $A(\mathbf{x})\colon \mathbb{R}^n \to \mathbb{R}^{n\times n}$ è una matrice non singolare

Sistemi non-lineari

- Al contrario del caso lineare non c'è una teoria generale che assicura l'esistenza e l'unicità della soluzione
- La risoluzione numerica è sempre basata su un algoritmo iterativo
- In generale, dato $\mathbf{x}^{(0)}$, la procedura iterativa consiste nel definire una successione $\mathbf{x}^{(k)}$

$$\mathbf{x}^{(k+1)} = \mathbf{G}(\mathbf{x}^{(k)}) \qquad k \geq 0$$

in cui l'operatore $\mathbf{G}\colon \mathbb{R}^n \to \mathbb{R}^n$ verifica

$$\mathbf{G}(\mathbf{x}^*) = \mathbf{x}^* \quad \text{ogni volta che} \quad \mathbf{F}(\mathbf{x}^*) = \mathbf{0}$$

- la scelta del punto di partenza $\mathbf{x}^{(0)}$ è molto importante per la convergenza

Esempio

$$\mathbf{G}(\mathbf{x}) = \mathbf{x} + A(\mathbf{x})\mathbf{F}(\mathbf{x})$$

in cui $A(\mathbf{x})\colon \mathbb{R}^n \to \mathbb{R}^{n\times n}$ è una matrice non singolare

Definizione di *contrazione*

Una funzione $\mathbf{G}\colon D \subset \mathbb{R}^n \to \mathbb{R}^n$ è una *contrazione* su un insieme $D_0 \subset D$ se esiste una costante $\alpha < 1$ tale che $||\mathbf{G}(\mathbf{x}) - \mathbf{G}(\mathbf{y})|| \leq \alpha ||\mathbf{x} - \mathbf{y}||$ per tutti gli $\mathbf{x}, \mathbf{y} \in D_0$ dove $||\cdot||$ è un'opportuna norma vettoriale.

Definizione di *contrazione*

Una funzione $\mathbf{G}\colon D \subset \mathbb{R}^n \to \mathbb{R}^n$ è una *contrazione* su un insieme $D_0 \subset D$ se esiste una costante $\alpha < 1$ tale che $||\mathbf{G}(\mathbf{x}) - \mathbf{G}(\mathbf{y})|| \leq \alpha ||\mathbf{x} - \mathbf{y}||$ per tutti gli $\mathbf{x}, \mathbf{y} \in D_0$ dove $||\cdot||$ è un'opportuna norma vettoriale.

Teorema delle contrazioni (o del punto fisso, o di Banach-Caccioppoli)

Supponiamo che $\mathbf{G}\colon D \subset \mathbb{R}^n \to \mathbb{R}^n$ è una contrazione su un insieme chiuso $D_0 \subset D$ e che $\mathbf{G}(\mathbf{x}) \in D_0$ per tutti gli $\mathbf{x} \in D_0$. Allora esiste un unico punto $\mathbf{x}^* \in D_0$ (chiamato *punto fisso* o *punto unito*) tale che $\mathbf{G}(\mathbf{x}^*) = \mathbf{x}^*$

Definizione di *contrazione*

Una funzione $\mathbf{G}\colon D \subset \mathbb{R}^n \to \mathbb{R}^n$ è una *contrazione* su un insieme $D_0 \subset D$ se esiste una costante $\alpha < 1$ tale che $||\mathbf{G}(\mathbf{x}) - \mathbf{G}(\mathbf{y})|| \leq \alpha ||\mathbf{x} - \mathbf{y}||$ per tutti gli $\mathbf{x}, \mathbf{y} \in D_0$ dove $||\cdot||$ è un'opportuna norma vettoriale.

Teorema delle contrazioni (o del punto fisso, o di Banach-Caccioppoli)

Supponiamo che $\mathbf{G}\colon D \subset \mathbb{R}^n \to \mathbb{R}^n$ è una contrazione su un insieme chiuso $D_0 \subset D$ e che $\mathbf{G}(\mathbf{x}) \in D_0$ per tutti gli $\mathbf{x} \in D_0$. Allora esiste un unico punto $\mathbf{x}^* \in D_0$ (chiamato *punto fisso* o *punto unito*) tale che $\mathbf{G}(\mathbf{x}^*) = \mathbf{x}^*$

- Il teorema delle contrazioni fornisce la convergenza globale dell'algoritmo iterativo $\mathbf{x}^{(k+1)} = \mathbf{G}(\mathbf{x}^{(k)})$ ma la definizione di contrattività è molto restrittiva
- Tipicamente non è possibile dimostrare la convergenza globale degli algoritmi comunemente usati

Teorema di convergenza locale (Ostrowski)

Supponiamo che:

- $\mathbf{G}\colon D \subset \mathbb{R}^n \to \mathbb{R}^n$ abbia un punto fisso $\mathbf{x}^*$ all'interno di D
- $\mathbf{G}$ sia differenziabile con continuità in un intorno di $\mathbf{x}^*$. Denotiamo con $J_{\mathbf{G}}$ la matrice Jacobiana di $\mathbf{G}$ e assumiamo che $\rho(J_{\mathbf{G}}(\mathbf{x}^*)) \leq 1$.

Allora:

- Esiste un intorno S di $\mathbf{x}^*$ tale che $S \subset D$
- Per ogni $\mathbf{x}^{(0)} \in S$ le iterate $\mathbf{x}^{(k+1)} = \mathbf{G}(\mathbf{x}^{(k)})$ $(k \geq 0)$ appartengono tutte a D e *convergono* a $\mathbf{x}^*$

Teorema di convergenza locale (Ostrowski)

Supponiamo che:

- $\mathbf{G}\colon D \subset \mathbb{R}^n \to \mathbb{R}^n$ abbia un punto fisso $\mathbf{x}^*$ all'interno di D
- $\mathbf{G}$ sia differenziabile con continuità in un intorno di $\mathbf{x}^*$. Denotiamo con $J_{\mathbf{G}}$ la matrice Jacobiana di $\mathbf{G}$ e assumiamo che $\rho(J_{\mathbf{G}}(\mathbf{x}^*)) \leq 1$.

Allora:

- Esiste un intorno S di $\mathbf{x}^*$ tale che $S \subset D$
- Per ogni $\mathbf{x}^{(0)} \in S$ le iterate $\mathbf{x}^{(k+1)} = \mathbf{G}(\mathbf{x}^{(k)})$ $(k \geq 0)$ appartengono tutte a D e *convergono* a $\mathbf{x}^*$

Teorema di Ostrowski nel caso monodimensionale

Sia $g\colon \mathbb{R} \to \mathbb{R}$ una funzione derivabile in un intorno di un punto fisso x^* e tale che $|g'(x^*)| < 1$. Allora esiste un $\varepsilon > 0$ tale che $\lim_{k\to\infty} x^{(k)} = x^*$ per ogni $x^{(0)}$ tale che $|x^* - x^{(0)}| < \varepsilon$

Definizione di ordine di convergenza

La successione $\{\mathbf{x}^{(k)}\}$ converge ad $\mathbf{x}^*$ con ordine $p \geq 1$ se:

$$\exists \beta > 0 \text{ tale che } \forall k \geq k_0 \text{ si ha } ||\mathbf{x}^{(k+1)} - \mathbf{x}^*|| \leq \beta ||\mathbf{x}^{(k)} - \mathbf{x}^*||^p$$

(nel caso in cui $p = 1$, per avere convergenza si deve anche richiedere che $\beta < 1$)

Definizione di ordine di convergenza

La successione $\{\mathbf{x}^{(k)}\}$ converge ad $\mathbf{x}^*$ con ordine $p \geq 1$ se:

$$\exists \beta > 0 \text{ tale che } \forall k \geq k_0 \text{ si ha } ||\mathbf{x}^{(k+1)} - \mathbf{x}^*|| \leq \beta ||\mathbf{x}^{(k)} - \mathbf{x}^*||^p$$

(nel caso in cui $p = 1$, per avere convergenza si deve anche richiedere che $\beta < 1$)

Teorema

Sia $g : D \subset \mathbb{R} \to \mathbb{R}$ la funzione che definisce l'iterazione $x^{(k+1)} = g(x^{(k)})$. Se

- $g \in C^{p+1}(I)$ $(p \geq 0)$ per un opportuno intorno I di un punto fisso x^*
- $g^{(i)}(x^*) = 0$ per $0 \leq i \leq p$
- $g^{(p+1)}(x^*) \neq 0$

Allora il metodo di punto fisso $x^{(k+1)} = g(x^{(k)})$ converge con ordine $p+1$ e

$$\lim_{x \to \infty} \frac{x^{(k+1)} - x^*}{(x^{(k)} - x^*)^p} = \frac{g^{(p+1)}(x^*)}{(p+1)!} \qquad p \geq 0$$

Esercizio

Quante iterazioni bisogna fare per guadagnare quindici cifre significative nell'errore (ovvero $\frac{||\mathbf{e}^{(k+r)}||}{||\mathbf{e}^{(k)}||} = \frac{||\mathbf{x}^{(k+r)} - \mathbf{x}^*||}{||\mathbf{x}^{(k)} - \mathbf{x}^*||} \lesssim 10^{-15}$)?

$$||\mathbf{e}^{(k+r)}|| \leq \beta ||\mathbf{e}^{(k+r-1)}||^p \leq \beta^{p+1} ||\mathbf{e}^{(k+r-2)}||^{p^2} \leq \cdots \leq \beta^{\sum_{i=0}^{r-1} p^i} ||\mathbf{e}^{(k)}||^{p^r}$$

ma $\sum_{i=0}^{r-1} p^i$ è uguale a r se $p = 1$ e a $\frac{p^r - 1}{p-1}$ se $p \neq 1$, perciò abbiamo

$$\frac{||\mathbf{e}^{(k+r)}||}{||\mathbf{e}^{(k)}||} \leq \begin{cases} \beta^r & \text{se } p = 1 \\ \left(\beta^{\frac{1}{p-1}} ||\mathbf{e}^{(k)}||\right)^{p^r - 1} & \text{se } p \neq 1 \end{cases}$$

ovvero

$$r \geq \begin{cases} \log\left(\frac{||\mathbf{e}^{(k+r)}||}{||\mathbf{e}^{(k)}||}\right) / \log \beta & \text{se } p = 1 \\ \log\left[\log\left(\frac{||\mathbf{e}^{(k+r)}||}{||\mathbf{e}^{(k)}||}\right) / \log\left(\beta^{\frac{1}{p-1}} ||\mathbf{e}^{(k)}||\right) + 1\right] / \log p & \text{se } p \neq 1 \end{cases}$$

$$r \geq \begin{cases} \log\left(\frac{||\mathbf{e}^{(k+r)}||}{||\mathbf{e}^{(k)}||}\right) / \log\beta & \text{se } p = 1 \\ \log\left[\log\left(\frac{||\mathbf{e}^{(k+r)}||}{||\mathbf{e}^{(k)}||}\right) / \log\left(\beta^{\frac{1}{p-1}} ||\mathbf{e}^{(k)}||\right) + 1\right] / \log p & \text{se } p \neq 1 \end{cases}$$

siccome vogliamo che $\frac{||\mathbf{e}^{(k+r)}||}{||\mathbf{e}^{(k)}||} \lesssim 10^{-15}$ otteniamo

$$r > \begin{cases} -\frac{15\log 10}{\log\beta} & \text{se } p = 1 \\ \log\left[1 - 15\log 10 / \log\left(\beta^{\frac{1}{p-1}} ||\mathbf{e}^{(k)}||\right)\right] / \log p & \text{se } p \neq 1 \end{cases}$$

supponendo che $\beta = 0.9$ e $||\mathbf{e}^{(k)}|| = 0.5$ otteniamo

$$r > \begin{cases} 328 & \text{se } p = 1 \\ 10 & \text{se } p = 1.5 \\ 6 & \text{se } p = 2 \\ 4 & \text{se } p = 3 \end{cases}$$

Metodo di bisezione (caso unidimensionale $f(x) = 0$)

- Si localizza lo zero di f individuando degli intervalli sempre più piccoli in cui questo zero è contenuto
- Se $f(x)$ è continua nell'intervallo $[a,b]$ e se $f(a)f(b) \leq 0$, allora $f(x)$ si annulla almeno una volta nell'intervallo $[a,b]$

Metodo di bisezione (caso unidimensionale $f(x) = 0$)

- Si localizza lo zero di f individuando degli intervalli sempre più piccoli in cui questo zero è contenuto
- Se $f(x)$ è continua nell'intervallo $[a, b]$ e se $f(a)f(b) \leq 0$, allora $f(x)$ si annulla almeno una volta nell'intervallo $[a, b]$

Algoritmo

Sia $a^{(0)} = a$, $b^{(0)} = b$ e $x^{(0)} = \frac{1}{2}(a^{(0)} + b^{(0)})$. Allora, per $k \geq 0$

1. Se $f(a^{(k)})f(x^{(k)}) \leq 0$ porre $a^{(k+1)} = a^{(k)}$, $b^{(k+1)} = x^{(k)}$, altrimenti porre $a^{(k+1)} = x^{(k)}$, $b^{(k+1)} = b^{(k)}$
2. Calcolare $x^{(k+1)} = \frac{1}{2}(a^{(k)} + b^{(k)})$

Metodo di bisezione (caso unidimensionale $f(x) = 0$)

- Si localizza lo zero di f individuando degli intervalli sempre più piccoli in cui questo zero è contenuto
- Se $f(x)$ è continua nell'intervallo $[a,b]$ e se $f(a)f(b) \leq 0$, allora $f(x)$ si annulla almeno una volta nell'intervallo $[a,b]$

Algoritmo

Sia $a^{(0)} = a$, $b^{(0)} = b$ e $x^{(0)} = \frac{1}{2}(a^{(0)} + b^{(0)})$. Allora, per $k \geq 0$

1. Se $f(a^{(k)})f(x^{(k)}) \leq 0$ porre $a^{(k+1)} = a^{(k)}$, $b^{(k+1)} = x^{(k)}$, altrimenti porre $a^{(k+1)} = x^{(k)}$, $b^{(k+1)} = b^{(k)}$
2. Calcolare $x^{(k+1)} = \frac{1}{2}(a^{(k)} + b^{(k)})$

- Questo algoritmo genera una successione di intervalli $I_k = [a_k, b_k]$ con $f(a_k)f(b_k) < 0$ e tali che $I_k \subset I_{k-1}$ con $|I_k| = |b_k - a_k| = \frac{1}{2}|b_{k-1} - a_{k-1}| = \frac{|I_k|}{2}$

Metodo di bisezione (caso unidimensionale $f(x) = 0$)

- Criteri di arresto:

$$|b_k - a_k| < \varepsilon_1 \qquad \text{oppure} \qquad \left|f\left(\frac{a_k + b_k}{2}\right)\right| \leq \varepsilon_2$$

Metodo di bisezione (caso unidimensionale $f(x) = 0$)

- Criteri di arresto:

$$|b_k - a_k| < \varepsilon_1 \qquad \text{oppure} \qquad \left|f\left(\frac{a_k + b_k}{2}\right)\right| \leq \varepsilon_2$$

- È un metodo globalmente convergente (cioé indipendente dal punto di partenza)
- Non garantisce una diminuzione monotona dell'errore
- È un algoritmo di convergenza sicura ma lenta

Metodo di bisezione (caso unidimensionale $f(x) = 0$)

- Criteri di arresto:

$$|b_k - a_k| < \varepsilon_1 \qquad \text{oppure} \qquad \left|f\left(\frac{a_k + b_k}{2}\right)\right| \leq \varepsilon_2$$

- È un metodo globalmente convergente (cioé indipendente dal punto di partenza)
- Non garantisce una diminuzione monotona dell'errore
- È un algoritmo di convergenza sicura ma lenta
- Utilizza come unica informazione i segni della funzione in alcuni punti

Metodo di bisezione (caso unidimensionale $f(x) = 0$)

- Criteri di arresto:

$$|b_k - a_k| < \varepsilon_1 \qquad \text{oppure} \qquad \left|f\left(\frac{a_k + b_k}{2}\right)\right| \leq \varepsilon_2$$

- È un metodo globalmente convergente (cioé indipendente dal punto di partenza)
- Non garantisce una diminuzione monotona dell'errore
- È un algoritmo di convergenza sicura ma lenta
- Utilizza come unica informazione i segni della funzione in alcuni punti $\Rightarrow$ Per una convergenza più veloce si devono degli algoritmi che tengano conto di informazioni aggiuntive (valori assunti dalla funzione, derivate, ...)

- Supponendo che $f \in C^1(\mathbb{R})$ e usando lo sviluppo di Taylor nell'intorno del punto x^* abbiamo che

$$f(x^*) = 0 = f(x) + (x^* - x)f'(\xi) \qquad \text{con } \xi \in]x^*, x[$$

- Supponendo che $f \in C^1(\mathbb{R})$ e usando lo sviluppo di Taylor nell'intorno del punto x^* abbiamo che

$$f(x^*) = 0 = f(x) + (x^* - x)f'(\xi) \qquad \text{con } \xi \in]x^*, x[$$

- Questo suggerisce la seguente famiglia di metodi iterativi

$$f(x^{(k)}) + (x^{(k+1)} - x^{(k)})q_k = 0 \qquad k \geq 0$$

ovvero

$$x^{(k+1)} = x^{(k)} - q_k^{-1} f(x^{(k)}) \qquad k \geq 0$$

in cui q_k è un'opportuna approssimazione di $f'(x^{(k)})$

- Supponendo che $f \in C^1(\mathbb{R})$ e usando lo sviluppo di Taylor nell'intorno del punto x^* abbiamo che

$$f(x^*) = 0 = f(x) + (x^* - x)f'(\xi) \qquad \text{con } \xi \in]x^*, x[$$

- Questo suggerisce la seguente famiglia di metodi iterativi

$$f(x^{(k)}) + (x^{(k+1)} - x^{(k)})q_k = 0 \qquad k \geq 0$$

ovvero

$$x^{(k+1)} = x^{(k)} - q_k^{-1} f(x^{(k)}) \qquad k \geq 0$$

in cui q_k è un'opportuna approssimazione di $f'(x^{(k)})$

- I diversi algoritmi si differenziano per come viene scelto q_k

Metodo delle corde: $q_k = q$

- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q} \qquad k \geq 0$$

in cui $q = \dfrac{f(b) - f(a)}{b - a}$ (pendenza della retta passante per i punti $(a, f(a))$ e $(b, f(b))$)

Metodo delle corde: $q_k = q$

- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q} \qquad k \geq 0$$

in cui $q = \dfrac{f(b) - f(a)}{b - a}$ (pendenza della retta passante per i punti $(a, f(a))$ e $(b, f(b))$)

- Il metodo converge linearmente se $0 < \dfrac{f'(x^*)}{q} < 2$

Metodo delle corde: $q_k = q$

- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q} \qquad k \geq 0$$

  in cui $q = \dfrac{f(b) - f(a)}{b - a}$ (pendenza della retta passante per i punti $(a, f(a))$ e $(b, f(b))$)

- Il metodo converge linearmente se $0 < \dfrac{f'(x^*)}{q} < 2$
- Una valutazione della funzione $f(x)$ per ogni iterazione

Metodo delle secanti

- Si scelgono due punti di partenza $x^{(-1)}$ e $x^{(0)}$
- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q_k} \qquad k \geq 0$$

in cui $q_k = \dfrac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}$

- q_k = pendenza della retta passante per i punti $(x^{(k-1)}, f(x^{(k-1)}))$ e $(x^{(k)}, f(x^{(k)}))$

Metodo delle secanti

- Si scelgono due punti di partenza $x^{(-1)}$ e $x^{(0)}$
- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q_k} \qquad k \geq 0$$

in cui $q_k = \dfrac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}$

- q_k = pendenza della retta passante per i punti $(x^{(k-1)}, f(x^{(k-1)}))$ e $(x^{(k)}, f(x^{(k)}))$
- Se $f \in C^2(I)$ (dove I è un opportuno intorno di x^*), $f''(x^*) \neq 0$ e $x^{(-1)}, x^{(0)} \in I$, allora il metodo converge con ordine $p = (1+\sqrt{5})/2 \simeq 1.63$

Metodo delle secanti

- Si scelgono due punti di partenza $x^{(-1)}$ e $x^{(0)}$
- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q_k} \qquad k \geq 0$$

in cui $q_k = \dfrac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}$

- q_k = pendenza della retta passante per i punti $(x^{(k-1)}, f(x^{(k-1)}))$ e $(x^{(k)}, f(x^{(k)}))$
- Se $f \in C^2(I)$ (dove I è un opportuno intorno di x^*), $f''(x^*) \neq 0$ e $x^{(-1)}, x^{(0)} \in I$, allora il metodo converge con ordine $p = (1+\sqrt{5})/2 \simeq 1.63$
- Due valutazione della funzione $f(x)$ per ogni iterazione

Regula falsi

È una variante del metodo delle secanti

- Si scelgono due punti di partenza $x^{(-1)}$ e $x^{(0)}$ tali che $f(x^{(-1)})f(x^{(0)}) < 0$
- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q_k} \qquad k \geq 0$$

in cui $q_k = \dfrac{f(x^{(k)}) - f(x^{(k')})}{x^{(k)} - x^{(k')}}$ dove k' è il massimo indice minore di k tale che $f(x^{(k)})f(x^{(k')}) < 0$

Regula falsi

È una variante del metodo delle secanti

- Si scelgono due punti di partenza $x^{(-1)}$ e $x^{(0)}$ tali che $f(x^{(-1)})f(x^{(0)}) < 0$
- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q_k} \qquad k \geq 0$$

  in cui $q_k = \dfrac{f(x^{(k)}) - f(x^{(k')})}{x^{(k)} - x^{(k')}}$ dove k' è il massimo indice minore di k tale che $f(x^{(k)})f(x^{(k')}) < 0$
- La convergenza di questo metodo è lineare
- Al contrario del metodo delle secanti, le iterate sono tutte contenute in $[x^{(-1)}, x^{(0)}]$ e il metodo è globalmente convergente

Regula falsi

È una variante del metodo delle secanti

- Si scelgono due punti di partenza $x^{(-1)}$ e $x^{(0)}$ tali che $f(x^{(-1)})f(x^{(0)}) < 0$
- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q_k} \qquad k \geq 0$$

  in cui $q_k = \dfrac{f(x^{(k)}) - f(x^{(k')})}{x^{(k)} - x^{(k')}}$ dove k' è il massimo indice minore di k tale che $f(x^{(k)})f(x^{(k')}) < 0$
- La convergenza di questo metodo è lineare
- Al contrario del metodo delle secanti, le iterate sono tutte contenute in $[x^{(-1)}, x^{(0)}]$ e il metodo è globalmente convergente
- Due valutazione della funzione $f(x)$ per ogni iterazione

Metodo di Newton

- Si richiede che f sia *derivabile*

Metodo di Newton

- Si richiede che f sia *derivabile*
- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q_k} \qquad k \geq 0$$

  in cui $q_k = f'(x^{(k)})$
- q_k è la pendenza della retta tangente alla funzione $f(x)$ nel punto $(x^{(k)}, f(x^{(k)}))$

Metodo di Newton

- Si richiede che f sia *derivabile*
- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q_k} \qquad k \geq 0$$

  in cui $q_k = f'(x^{(k)})$
- q_k è la pendenza della retta tangente alla funzione $f(x)$ nel punto $(x^{(k)}, f(x^{(k)}))$
- La convergenza di questo metodo è quadratica vicino alla soluzione

Metodo di Newton

- Si richiede che f sia *derivabile*
- Si costruisce una successione

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q_k} \qquad k \geq 0$$

  in cui $q_k = f'(x^{(k)})$
- q_k è la pendenza della retta tangente alla funzione $f(x)$ nel punto $(x^{(k)}, f(x^{(k)}))$
- La convergenza di questo metodo è quadratica vicino alla soluzione
- Per ogni iterazione bisogna fare una valutazione di $f(x)$ e una valutazione della derivata $f'(x)$

Metodo di Newton

Metodo di Newton

Sia $\mathbf{F}\colon \mathbb{R}^n \to \mathbb{R}^n$ una funzione differenziabile, e sia $J_{\mathbf{F}}(\mathbf{x})$ la sua matrice Jacobiana

$$(J_{\mathbf{F}}(\mathbf{x}))_{ij} = \left(\frac{\partial F_i}{\partial x_j}\right)(\mathbf{x})$$

Metodo di Newton

Metodo di Newton

Sia $\mathbf{F}\colon \mathbb{R}^n \to \mathbb{R}^n$ una funzione differenziabile, e sia $J_{\mathbf{F}}(\mathbf{x})$ la sua matrice Jacobiana

$$(J_{\mathbf{F}}(\mathbf{x}))_{ij} = \left(\frac{\partial F_i}{\partial x_j}\right)(\mathbf{x})$$

Il metodo di Newton si scrive allora come segue:

Dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \ldots,$ fino alla convergenza

1. Risolvere il sistema lineare $J_{\mathbf{F}}(\mathbf{x}^{(k)})\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{F}(\mathbf{x}^{(k)})$
2. Calcolare $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)}$

Metodo di Newton

Metodo di Newton

Sia $\mathbf{F}\colon \mathbb{R}^n \to \mathbb{R}^n$ una funzione differenziabile, e sia $J_{\mathbf{F}}(\mathbf{x})$ la sua matrice Jacobiana

$$(J_{\mathbf{F}}(\mathbf{x}))_{ij} = \left(\frac{\partial F_i}{\partial x_j}\right)(\mathbf{x})$$

Il metodo di Newton si scrive allora come segue:

Dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \ldots$, fino alla convergenza

1. Risolvere il sistema lineare $J_{\mathbf{F}}(\mathbf{x}^{(k)})\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{F}(\mathbf{x}^{(k)})$
2. Calcolare $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)}$

- Il metodo di Newton è un modo di risolvere un problema non lineare mediante la linearizzazione

Metodo di Newton

Metodo di Newton

Sia $\mathbf{F}\colon \mathbb{R}^n \to \mathbb{R}^n$ una funzione differenziabile, e sia $J_{\mathbf{F}}(\mathbf{x})$ la sua matrice Jacobiana

$$(J_{\mathbf{F}}(\mathbf{x}))_{ij} = \left(\frac{\partial F_i}{\partial x_j}\right)(\mathbf{x})$$

Il metodo di Newton si scrive allora come segue:

Dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \ldots$, fino alla convergenza

1. Risolvere il sistema lineare $J_{\mathbf{F}}(\mathbf{x}^{(k)})\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{F}(\mathbf{x}^{(k)})$
2. Calcolare $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)}$

- Il metodo di Newton è un modo di risolvere un problema non lineare mediante la linearizzazione
- E' richiesta la risoluzione di un sistema lineare per ogni passo di iterazione

Metodo di Newton

Risultato di convergenza

Se:

- $\mathbf{F}: \mathbb{R}^n \to \mathbb{R}^n$ una funzione differenziabile in un aperto convesso di $\mathbb{R}^n$ contenente $\mathbf{x}^*$
- $J_{\mathbf{F}}(\mathbf{x}^*)$ è invertibile e tale che $||J_{\mathbf{F}}^{-1}(\mathbf{x}^*)|| \leq C$, con $C > 0$
- esistono due costanti positive R e L tali che

$$||J_{\mathbf{F}}(\mathbf{x}) - J_{\mathbf{F}}(\mathbf{y})|| \leq L||\mathbf{x} - \mathbf{y}|| \qquad \forall \mathbf{x}, \mathbf{y} \in B(\mathbf{x}^*; R)$$

Allora esiste $r > 0$ tale che, per ogni $\mathbf{x}^{(0)} \in B(\mathbf{x}^*; R)$, la sequenza

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J_{\mathbf{F}}^{-1}(\mathbf{x}^{(k)})\mathbf{F}(\mathbf{x}^{(k)})$$

converge a $\mathbf{x}^*$ con

$$||\mathbf{x}^{(k+1)} - \mathbf{x}^*|| \leq CL||\mathbf{x}^{(k)} - \mathbf{x}^*||^2$$

Metodo di Newton

Considerazioni

- Come risolvere $J_{\mathbf{F}}(\mathbf{x}^{(k)})\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{F}(\mathbf{x}^{(k)})$?

Metodo di Newton

Considerazioni

- Come risolvere $J_{\mathbf{F}}(\mathbf{x}^{(k)})\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{F}(\mathbf{x}^{(k)})$?
  - Calcolare (e memorizzare) la matrice Jacobiana $J_{\mathbf{F}}(\mathbf{x}^{(k)})$ e poi risolvere il sistema lineare
  - Calcolare il prodotto $J_{\mathbf{F}}(\mathbf{x}^{(k)})\mathbf{y}$ per un generico $\mathbf{y} \in \mathbb{R}^n$ ed utilizzare un metodo di risoluzione *matrix-free* (per esempio GMRES)

Metodo di Newton

Considerazioni

- Come risolvere $J_{\mathbf{F}}(\mathbf{x}^{(k)})\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{F}(\mathbf{x}^{(k)})$?
  - Calcolare (e memorizzare) la matrice Jacobiana $J_{\mathbf{F}}(\mathbf{x}^{(k)})$ e poi risolvere il sistema lineare
  - Calcolare il prodotto $J_{\mathbf{F}}(\mathbf{x}^{(k)})\mathbf{y}$ per un generico $\mathbf{y} \in \mathbb{R}^n$ ed utilizzare un metodo di risoluzione *matrix-free* (per esempio GMRES)
- In generale, se n è grande il calcolo di $J_{\mathbf{F}}(\mathbf{x}^{(k)})$ può essere costoso
- Inoltre, $J_{\mathbf{F}}(\mathbf{x}^{(k)})$ può essere mal condizionata
- $\Rightarrow$ varianti al metodo di Newton

Varianti del Metodo di Newton

Valutazione ciclica della matrice Jacobiana

- Si usa la stessa matrice Jacobiana per un certo numero r di iterazioni
- Ha senso usare questo metodo solo se si utilizzano approcci basati sulla fattorizzazione (eseguita una volta ogni r iterazioni)

Varianti del Metodo di Newton

Valutazione ciclica della matrice Jacobiana

- Si usa la stessa matrice Jacobiana per un certo numero r di iterazioni
- Ha senso usare questo metodo solo se si utilizzano approcci basati sulla fattorizzazione (eseguita una volta ogni r iterazioni)

Algoritmo

Per $x^{(0)} \in \mathbb{R}^n$ e $k = 0, 1, \ldots$ fino alla convergenza

- Calcolare (e fattorizzare) $A = J_{\mathbf{F}}(\mathbf{x}^{(rk)})$
- Per $i = 0, \ldots, r-1$
  - Risolvere $A\boldsymbol{\delta}\mathbf{x}^{(rk+i)} = -\mathbf{F}(\mathbf{x}^{(rk+i)})$
  - Calcolare $\mathbf{x}^{(rk+i+1)} = \mathbf{x}^{(rk+i)} + \boldsymbol{\delta}\mathbf{x}^{(rk+i)}$

Risoluzione approssimata del sistema lineare

- Si risolve il sistema lineare con un metodo iterativo fissando *a priori* il numero massimo di iterazioni o il test di convergenza viene effettuato con una soglia abbastanza alta
- Questi schemi vengono chiamati Newton-Jacobi, Newton-SOR, Newton-Richardson, Newton-Krylov, ...

Risoluzione approssimata del sistema lineare

- Si risolve il sistema lineare con un metodo iterativo fissando *a priori* il numero massimo di iterazioni o il test di convergenza viene effettuato con una soglia abbastanza alta
- Questi schemi vengono chiamati Newton-Jacobi, Newton-SOR, Newton-Richardson, Newton-Krylov, ...

Esempio: Newton-Gauss-Seidel a un passo

- Il singolo passo dell'iterazione di Gauss-Seidel è $\mathbf{x}^{(r+1)} = (D-E)^{-1}F\mathbf{x}^{(r)} + (D-E)^{-1}\mathbf{b}$

Risoluzione approssimata del sistema lineare

- Si risolve il sistema lineare con un metodo iterativo fissando *a priori* il numero massimo di iterazioni o il test di convergenza viene effettuato con una soglia abbastanza alta
- Questi schemi vengono chiamati Newton-Jacobi, Newton-SOR, Newton-Richardson, Newton-Krylov, ...

Esempio: Newton-Gauss-Seidel a un passo

- Il singolo passo dell'iterazione di Gauss-Seidel è $\mathbf{x}^{(r+1)} = (D-E)^{-1}F\mathbf{x}^{(r)} + (D-E)^{-1}\mathbf{b}$
- Il singolo passo dell'iterazione di Newton è $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J_{\mathbf{F}}^{-1}(\mathbf{x}^{(k)})\mathbf{F}(\mathbf{x}^{(k)})$

Risoluzione approssimata del sistema lineare

- Si risolve il sistema lineare con un metodo iterativo fissando *a priori* il numero massimo di iterazioni o il test di convergenza viene effettuato con una soglia abbastanza alta
- Questi schemi vengono chiamati Newton-Jacobi, Newton-SOR, Newton-Richardson, Newton-Krylov, ...

Esempio: Newton-Gauss-Seidel a un passo

- Il singolo passo dell'iterazione di Gauss-Seidel è $\mathbf{x}^{(r+1)} = (D-E)^{-1}F\mathbf{x}^{(r)} + (D-E)^{-1}\mathbf{b}$
- Il singolo passo dell'iterazione di Newton è $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J_{\mathbf{F}}^{-1}(\mathbf{x}^{(k)})\mathbf{F}(\mathbf{x}^{(k)})$
- Il termine noto del sistema lineare $J_{\mathbf{F}}(\mathbf{x}^{(k)})\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{F}(\mathbf{x}^{(k)})$ è $\mathbf{b} = -\mathbf{F}(\mathbf{x}^{(k)})$

Risoluzione approssimata del sistema lineare

- Si risolve il sistema lineare con un metodo iterativo fissando *a priori* il numero massimo di iterazioni o il test di convergenza viene effettuato con una soglia abbastanza alta
- Questi schemi vengono chiamati Newton-Jacobi, Newton-SOR, Newton-Richardson, Newton-Krylov, ...

Esempio: Newton-Gauss-Seidel a un passo

- Il singolo passo dell'iterazione di Gauss-Seidel è $\mathbf{x}^{(r+1)} = (D-E)^{-1}F\mathbf{x}^{(r)} + (D-E)^{-1}\mathbf{b}$
- Il singolo passo dell'iterazione di Newton è $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J_{\mathbf{F}}^{-1}(\mathbf{x}^{(k)})\mathbf{F}(\mathbf{x}^{(k)})$
- Il termine noto del sistema lineare $J_{\mathbf{F}}(\mathbf{x}^{(k)})\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{F}(\mathbf{x}^{(k)})$ è $\mathbf{b} = -\mathbf{F}(\mathbf{x}^{(k)})$
- Si effettua un passo del metodo di Gauss-Seidel per calcolare $\boldsymbol{\delta}\mathbf{x}^{(k)}$

$$\boldsymbol{\delta}\mathbf{x}_1^{(k)} = -J_{\mathbf{F}}^{-1}(\mathbf{x}^{(k)})\mathbf{F}(\mathbf{x}^{(k)}) \approx (D_k - E_k)^{-1}F_k\boldsymbol{\delta}\mathbf{x}_0^{(k)} - (D_k - E_k)^{-1}\mathbf{F}(\mathbf{x}^{(k)})$$

Risoluzione approssimata del sistema lineare

- Si risolve il sistema lineare con un metodo iterativo fissando *a priori* il numero massimo di iterazioni o il test di convergenza viene effettuato con una soglia abbastanza alta
- Questi schemi vengono chiamati Newton-Jacobi, Newton-SOR, Newton-Richardson, Newton-Krylov, ...

Esempio: Newton-Gauss-Seidel a un passo

- Il singolo passo dell'iterazione di Gauss-Seidel è $\mathbf{x}^{(r+1)} = (D-E)^{-1}F\mathbf{x}^{(r)} + (D-E)^{-1}\mathbf{b}$
- Il singolo passo dell'iterazione di Newton è $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J_{\mathbf{F}}^{-1}(\mathbf{x}^{(k)})\mathbf{F}(\mathbf{x}^{(k)})$
- Il termine noto del sistema lineare $J_{\mathbf{F}}(\mathbf{x}^{(k)})\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{F}(\mathbf{x}^{(k)})$ è $\mathbf{b} = -\mathbf{F}(\mathbf{x}^{(k)})$
- Si effettua un passo del metodo di Gauss-Seidel per calcolare $\boldsymbol{\delta}\mathbf{x}^{(k)}$

$$\boldsymbol{\delta}\mathbf{x}_1^{(k)} = -J_{\mathbf{F}}^{-1}(\mathbf{x}^{(k)})\mathbf{F}(\mathbf{x}^{(k)}) \approx (D_k - E_k)^{-1}F_k\boldsymbol{\delta}\mathbf{x}_0^{(k)} - (D_k - E_k)^{-1}\mathbf{F}(\mathbf{x}^{(k)})$$

- Ma $\boldsymbol{\delta}\mathbf{x}_0^{(k)} = \mathbf{0}$ e quindi $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (D_k - E_k)^{-1}\mathbf{F}(\mathbf{x}^{(k)})$

Metodi Quasi-Newton

Generalizzazione del metodo di Newton

Algoritmo

Sia $\mathbf{F}\colon \mathbb{R}^n \to \mathbb{R}^n$ una funzione differenziabile, e sia $\mathbf{x}^{(0)} \in \mathbb{R}$ un punto iniziale. Ad ogni passo k si effettuano le seguenti operazioni

- calcolare $\mathbf{F}(\mathbf{x}^{(k)})$
- scegliere $\widetilde{J}_{\mathbf{F}}(\mathbf{x}^{(k)})$ come la matrice Jacobiana esatta $J_{\mathbf{F}}(\mathbf{x}^{(k)})$ o una sua approssimazione
- risolvere il sistema lineare $\widetilde{J}_{\mathbf{F}}(\mathbf{x}^{(k)})\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{F}(\mathbf{x}^{(k)})$
- porre $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \boldsymbol{\delta}\mathbf{x}^{(k)}$ dove α_k è un parametro opportuno

Metodo di Broyden

- Generalizzazione del metodo delle secanti nel caso multidimensionale
- Dati due vettori $\mathbf{x}^0$ e $\mathbf{x}^1$ si risolve

$$Q_k \boldsymbol{\delta}\mathbf{x}^{(k)} = Q_k\left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right) = -\mathbf{F}(\mathbf{x}^{(k)}) \quad k \geq 1$$

- Per ogni passo k si cerca una matrice $Q_k \in \mathbb{R}^{n\times n}$ tale che

$$Q_k\left(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right) = \mathbf{F}(\mathbf{x}^{(k)}) - \mathbf{F}(\mathbf{x}^{(k-1)}) = \mathbf{b}^{(k-1)} \qquad k \geq 1$$

  ma questo è un sistema indeterminato (n equazioni per n^2 variabili)
- Per determinare univocamente Q_k si richiede che valga

$$Q_k\left(\mathbf{x}^{(k)} - \mathbf{x}^{(k-j)}\right) = \mathbf{F}(\mathbf{x}^{(k)}) - \mathbf{F}(\mathbf{x}^{(k-j)}) \qquad j = 1,\ldots,n \tag{2}$$

  con $\mathbf{x}^{(k-j)},\ldots,\mathbf{x}^{(k)}$ linearmente indipendenti
- In pratica $\mathbf{x}^{(k-j)},\ldots,\mathbf{x}^{(k)}$ tendono a diventare linearmente dipendenti e il metodo precedente non è stabile

Metodo di Broyden

- Si cerca allora Q_k in modo che venga minimizzata la differenza tra le approssimazioni di $\mathbf{F}(\mathbf{x})$:

$$\mathbf{F}(\mathbf{x}^{(k)}) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) \quad \text{e} \quad \mathbf{F}(\mathbf{x}^{(k-1)}) + Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)})$$

Metodo di Broyden

- Si cerca allora Q_k in modo che venga minimizzata la differenza tra le approssimazioni di $\mathbf{F}(\mathbf{x})$:

$$\mathbf{F}(\mathbf{x}^{(k)}) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) \quad \text{e} \quad \mathbf{F}(\mathbf{x}^{(k-1)}) + Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)})$$

$$\begin{aligned}
\mathbf{d}_k &= \mathbf{F}(\mathbf{x}^{(k)}) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) - \mathbf{F}(\mathbf{x}^{(k-1)}) - Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)}) \\
&= \left(\mathbf{F}(\mathbf{x}^{(k)}) - \mathbf{F}(\mathbf{x}^{(k-1)})\right) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) - Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)}) \\
&= Q_k\left(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) - Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)}) \\
&= (Q_k - Q_{k-1})(\mathbf{x} - \mathbf{x}^{(k-1)})
\end{aligned}$$

Metodo di Broyden

- Si cerca allora Q_k in modo che venga minimizzata la differenza tra le approssimazioni di $\mathbf{F}(\mathbf{x})$:

$$\mathbf{F}(\mathbf{x}^{(k)}) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) \quad \text{e} \quad \mathbf{F}(\mathbf{x}^{(k-1)}) + Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)})$$

$$\begin{aligned}
\mathbf{d}_k &= \mathbf{F}(\mathbf{x}^{(k)}) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) - \mathbf{F}(\mathbf{x}^{(k-1)}) - Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)}) \\
&= \left(\mathbf{F}(\mathbf{x}^{(k)}) - \mathbf{F}(\mathbf{x}^{(k-1)})\right) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) - Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)}) \\
&= Q_k\left(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) - Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)}) \\
&= (Q_k - Q_{k-1})(\mathbf{x} - \mathbf{x}^{(k-1)})
\end{aligned}$$

- Scriviamo $\mathbf{x} - \mathbf{x}^{(k-1)} = \alpha \boldsymbol{\delta}\mathbf{x}^{(k-1)} + \mathbf{s}$ dove $\langle \boldsymbol{\delta}\mathbf{x}^{(k-1)}, \mathbf{s} \rangle = 0$

Metodo di Broyden

- Si cerca allora Q_k in modo che venga minimizzata la differenza tra le approssimazioni di $\mathbf{F}(\mathbf{x})$:

$$\mathbf{F}(\mathbf{x}^{(k)}) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) \quad \text{e} \quad \mathbf{F}(\mathbf{x}^{(k-1)}) + Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)})$$

$$\begin{aligned}
\mathbf{d}_k &= \mathbf{F}(\mathbf{x}^{(k)}) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) - \mathbf{F}(\mathbf{x}^{(k-1)}) - Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)}) \\
&= \left(\mathbf{F}(\mathbf{x}^{(k)}) - \mathbf{F}(\mathbf{x}^{(k-1)})\right) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) - Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)}) \\
&= Q_k\left(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right) + Q_k(\mathbf{x} - \mathbf{x}^{(k)}) - Q_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)}) \\
&= (Q_k - Q_{k-1})(\mathbf{x} - \mathbf{x}^{(k-1)})
\end{aligned}$$

- Scriviamo $\mathbf{x} - \mathbf{x}^{(k-1)} = \alpha \boldsymbol{\delta}\mathbf{x}^{(k-1)} + \mathbf{s}$ dove $\langle \boldsymbol{\delta}\mathbf{x}^{(k-1)}, \mathbf{s} \rangle = 0$

$$\mathbf{d}_k = (Q_k - Q_{k-1})\left(\alpha \boldsymbol{\delta}\mathbf{x}^{(k-1)} + \mathbf{s}\right) = \alpha\left(\mathbf{b}^{(k-1)} - Q_{k-1}\boldsymbol{\delta}\mathbf{x}^{(k-1)}\right) + (Q_k - Q_{k-1})\mathbf{s}$$

Metodo di Broyden

- Quindi la minimizzazione deve essere effettuata solo sul termine $(Q_k - Q_{k-1})\mathbf{s}$ (gli altri non dipendono da Q_k)

Metodo di Broyden

- Quindi la minimizzazione deve essere effettuata solo sul termine $(Q_k - Q_{k-1})\mathbf{s}$ (gli altri non dipendono da Q_k)
- Il problema è diventato: trovare la matrice Q_k tale che $||(Q_k - Q_{k-1})\mathbf{s}||_2$ è minima per ogni $\mathbf{s}$ ortogonale a $\boldsymbol{\delta}\mathbf{x}^{(k-1)}$ e con la condizione Eq. (2)

Metodo di Broyden

- Quindi la minimizzazione deve essere effettuata solo sul termine $(Q_k - Q_{k-1})\mathbf{s}$ (gli altri non dipendono da Q_k)
- Il problema è diventato: trovare la matrice Q_k tale che $||(Q_k - Q_{k-1})\mathbf{s}||_2$ è minima per ogni $\mathbf{s}$ ortogonale a $\boldsymbol{\delta}\mathbf{x}^{(k-1)}$ e con la condizione Eq. (2)
- Si può dimostrare che tale matrice esiste e può essere calcolata per ricorrenza come

$$Q_k = Q_{k-1} + \frac{(\mathbf{b}^{(k-1)} - Q_{k-1}\boldsymbol{\delta}\mathbf{x}^{(k-1)})\boldsymbol{\delta}\mathbf{x}^{(k-1)^T}}{||\boldsymbol{\delta}\mathbf{x}^{(k-1)}||_2^2}$$

Metodo di Broyden

- Quindi la minimizzazione deve essere effettuata solo sul termine $(Q_k - Q_{k-1})\mathbf{s}$ (gli altri non dipendono da Q_k)
- Il problema è diventato: trovare la matrice Q_k tale che $||(Q_k - Q_{k-1})\mathbf{s}||_2$ è minima per ogni $\mathbf{s}$ ortogonale a $\boldsymbol{\delta}\mathbf{x}^{(k-1)}$ e con la condizione Eq. (2)
- Si può dimostrare che tale matrice esiste e può essere calcolata per ricorrenza come

$$Q_k = Q_{k-1} + \frac{(\mathbf{b}^{(k-1)} - Q_{k-1}\boldsymbol{\delta}\mathbf{x}^{(k-1)})\boldsymbol{\delta}\mathbf{x}^{(k-1)^T}}{||\boldsymbol{\delta}\mathbf{x}^{(k-1)}||_2^2}$$

- L'inizializzazione è effettuata utilizzando la matrice Jacobiana esatta $Q_0 = J_{\mathbf{F}}(\mathbf{x}^{(0)})$

Metodo di Broyden

- Quindi la minimizzazione deve essere effettuata solo sul termine $(Q_k - Q_{k-1})\mathbf{s}$ (gli altri non dipendono da Q_k)
- Il problema è diventato: trovare la matrice Q_k tale che $||(Q_k - Q_{k-1})\mathbf{s}||_2$ è minima per ogni $\mathbf{s}$ ortogonale a $\boldsymbol{\delta}\mathbf{x}^{(k-1)}$ e con la condizione Eq. (2)
- Si può dimostrare che tale matrice esiste e può essere calcolata per ricorrenza come

$$Q_k = Q_{k-1} + \frac{(\mathbf{b}^{(k-1)} - Q_{k-1}\boldsymbol{\delta}\mathbf{x}^{(k-1)})\boldsymbol{\delta}\mathbf{x}^{(k-1)^T}}{||\boldsymbol{\delta}\mathbf{x}^{(k-1)}||_2^2}$$

- L'inizializzazione è effettuata utilizzando la matrice Jacobiana esatta $Q_0 = J_{\mathbf{F}}(\mathbf{x}^{(0)})$
- Se $\mathbf{x}^{(0)}$ è abbastanza vicino a $\mathbf{x}^*$, il metodo di Broyden converge a $\mathbf{x}^*$ in maniera *superlineare*, ovvero

$$||\mathbf{x}^{(k)} - \mathbf{x}^*|| \leq c_k||\mathbf{x}^{(k-1)} - \mathbf{x}^*|| \qquad \text{dove} \quad \lim_{k\to\infty} c_k = 0$$

- I metodi Quasi-Newton $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \widetilde{J}_{\mathbf{F}}(\mathbf{x}^{(k)})^{-1}\mathbf{F}(\mathbf{x}^{(k)})$ possono essere visti come metodi di punto fisso con funzione di iterazione

$$\mathbf{G}_N(\mathbf{x}) = \mathbf{x} - \alpha_k \widetilde{J}_{\mathbf{F}}(\mathbf{x})^{-1}\mathbf{F}(\mathbf{x})$$

- I metodi Quasi-Newton $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \widetilde{J}_{\mathbf{F}}(\mathbf{x}^{(k)})^{-1}\mathbf{F}(\mathbf{x}^{(k)})$ possono essere visti come metodi di punto fisso con funzione di iterazione

$$\mathbf{G}_N(\mathbf{x}) = \mathbf{x} - \alpha_k \widetilde{J}_{\mathbf{F}}(\mathbf{x})^{-1}\mathbf{F}(\mathbf{x})$$

- Se denotiamo $\mathbf{r}^{(k)} = \mathbf{F}(\mathbf{x}^{(k)})$ il residuo al passo k, possiamo anche scrivere il metodo Quasi-Newton come

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \widetilde{J}_{\mathbf{F}}(\mathbf{x}^{(k)})^{-1}\mathbf{r}^{(k)}$$

ovvero come un metodo di Richardson precondizionato, dove la matrice di precondizionamento è

$$P_k = -\widetilde{J}_{\mathbf{F}}(\mathbf{x}^{(k)})$$

Outline

Domanda

Dato $\mathbf{x}^{(k)}$, come calcolare la matrice Jacobiana $J_{\mathbf{F}}(\mathbf{x}^{(k)})$ o una sua approssimazione $\widetilde{J}_{\mathbf{F}}(\mathbf{x}^{(k)})$?

Domanda

Dato $\mathbf{x}^{(k)}$, come calcolare la matrice Jacobiana $J_{\mathbf{F}}(\mathbf{x}^{(k)})$ o una sua approssimazione $\widetilde{J}_{\mathbf{F}}(\mathbf{x}^{(k)})$?

Metodi per calcolare numericamente le derivate

- Differenziazione analitica seguita da discretizzazione
- Differenze divise
- Metodo della variabile complessa
- Differenziazione automatica

Domanda

Dato $\mathbf{x}^{(k)}$, come calcolare la matrice Jacobiana $J_{\mathbf{F}}(\mathbf{x}^{(k)})$ o una sua approssimazione $\widetilde{J}_{\mathbf{F}}(\mathbf{x}^{(k)})$?

Metodi per calcolare numericamente le derivate

- Differenziazione analitica seguita da discretizzazione
- Differenze divise
- Metodo della variabile complessa
- Differenziazione automatica

Nota:

- Se n non è troppo grande ed abbiamo un metodo (programma) per calcolare il prodotto $A\mathbf{x}$ dove A è una matrice $n \times n$ (sconosciuta o non disponibile) e $\mathbf{x} \in \mathbb{R}^n$, possiamo costruire A colonna per colonna calcolando $A\mathbf{e}_i$ $(i = 1, \ldots, n)$ dove $\mathbf{e}_i$ è l'i-esimo vettore della base canonica di $\mathbb{R}^n$

Domanda

Dato $\mathbf{x}^{(k)}$, come calcolare la matrice Jacobiana $J_{\mathbf{F}}(\mathbf{x}^{(k)})$ o una sua approssimazione $\widetilde{J}_{\mathbf{F}}(\mathbf{x}^{(k)})$?

Metodi per calcolare numericamente le derivate

- Differenziazione analitica seguita da discretizzazione
- Differenze divise
- Metodo della variabile complessa
- Differenziazione automatica

Nota:

- Se n non è troppo grande ed abbiamo un metodo (programma) per calcolare il prodotto $A\mathbf{x}$ dove A è una matrice $n \times n$ (sconosciuta o non disponibile) e $\mathbf{x} \in \mathbb{R}^n$, possiamo costruire A colonna per colonna calcolando $A\mathbf{e}_i$ $(i = 1, \ldots, n)$ dove $\mathbf{e}_i$ è l'i-esimo vettore della base canonica di $\mathbb{R}^n$
- Se A è sparsa, esistono delle tecniche di compressione che permettono di calcolare la matrice effettuando un numero di moltiplicazioni matrice per vettore $< n$

So do you need derivatives?

Given a program **P** computing a function Φ

$$\begin{aligned} \Phi\colon \mathbb{R}^m &\longrightarrow \mathbb{R}^n \\ x &\longmapsto y = \Phi(x) \end{aligned}$$

we want to build a program that computes the derivatives of Φ.

Specifically, we want the derivatives of the dependent, i.e. some variables in $y \in \mathbb{R}^n$, with respect to the independent, i.e. some variables in $x \in \mathbb{R}^m$.

Which derivative do you want?

Many choices are possible:

- Jacobian matrices: $J = \left(\frac{d\Phi}{dx} \right) \in \mathbb{R}^{n,m}$
- Directional or tangential derivatives (differentials): $\delta y = \left(\frac{d\Phi}{dx} \right) \delta x$
- Gradients
- High-order derivative tensors (Hessian, . . .)
- . . .

Differentiate-then-discretize approach

- Given the equation $y = \Phi(x)$ we can write a new set of equations, whose solutions are the derivative of the initial result
- $\Longrightarrow$ Discretization of the new equations
- Write a new program **P'** that solves these new equations for the desired derivatives

Differentiate-then-discretize approach

- Given the equation $y = \Phi(x)$ we can write a new set of equations, whose solutions are the derivative of the initial result
- $\Longrightarrow$ Discretization of the new equations
- Write a new program **P′** that solves these new equations for the desired derivatives
- Pros:
  - mathematically “elegant”
- Cons:
  - Hard to develop a new program
  - Possible inconsistencies between discretization grids
  - Error prone
  - Sometimes the equation $y = \Phi(x)$ is not available

Divided differences

$$\left(\frac{d\Phi}{dx}\right)\delta x = \begin{cases} \dfrac{\Phi(x+\varepsilon\delta x)-\Phi(x)}{\varepsilon} + O(\varepsilon) \\ \dfrac{\Phi(x+\varepsilon\delta x)-\Phi(x-\varepsilon\delta x)}{2\varepsilon} + O(\varepsilon^2) \end{cases}$$

Divided differences

$$\left(\frac{d\Phi}{dx}\right)\delta x = \begin{cases} \dfrac{\Phi(x+\varepsilon\delta x)-\Phi(x)}{\varepsilon} + O(\varepsilon) \\ \dfrac{\Phi(x+\varepsilon\delta x)-\Phi(x-\varepsilon\delta x)}{2\varepsilon} + O(\varepsilon^2) \end{cases}$$

- Pros:
  - Very easy to implement
  - Requires only the results of the original program **P**
  - “Black-box” approach (the implementation of **P** is not needed)
- Cons:
  - **Only directional derivatives**
  - **Expensive (extra evaluations of the function for each direction)**
  - Problems on the choice of the stepsize ε (tradeoff between approximation and roundoff errors)
  - Approximate derivatives

Complex variables method

- Taylor expansion of Φ using a complex increment $i\varepsilon\delta x$

$$\left(\frac{d\Phi}{dx}\right)\Bigg|_x \delta x = \frac{\text{Im}\left[\Phi(x+i\varepsilon\delta x)\right]}{\varepsilon} + O(\varepsilon^2)$$

Complex variables method

- Taylor expansion of Φ using a complex increment $i\varepsilon\delta x$

$$\left(\frac{d\Phi}{dx}\right)\Bigg|_x \delta x = \frac{\text{Im}\left[\Phi(x+i\varepsilon\delta x)\right]}{\varepsilon} + O(\varepsilon^2)$$

- Pros:
  - Very small ε can be used $\Rightarrow$ low errors
- Cons:
  - Only directional derivatives
  - Work from the user to modify the program (from real-values functions to complex-variables functions)

Automatic Differentiation (AD)

Augment the program **P** so that it computes the analytic derivatives

```
P: a = b*T(10) + c
```

The differentiated program must somehow compute

```
P': da = db*T(10) + b*dT(10) + dc
```

How we can achieve this?

- AD by overloading
- AD by program (source) transformation

Automatic Differentiation (AD)

Augment the program **P** so that it computes the analytic derivatives

```
P: a = b*T(10) + c
```

The differentiated program must somehow compute

```
P': da = db*T(10) + b*dT(10) + dc
```

How we can achieve this?

- AD by overloading
- AD by program (source) transformation

AD gives us exact differentiation!

AD by overloading

Tools: ADOL-C, ADC, ADF, FADBAD++,...

Few manipulations required:

- new type definitions: `DOUBLE` $\rightarrow$ `ADOUBLE` ;
- link with provided overloaded `+`, `-`, `*`,...

Easy extension to higher-order, Taylor series, intervals, ... but not so easy for gradients.

AD by Program transformation

Tools: TAPENADE, ADIFOR, ADIC, OPENAD,

...

Complex transformation required:

- Build a new program that computes the analytic derivatives explicitly.
- Requires a compiler-like, sophisticated tool
  1. PARSING,
  2. ANALYSIS,
  3. DIFFERENTIATION,
  4. REGENERATION

Overloading *vs* Transformation

- Overloading is versatile:
  - easy to use;
  - simple to extend to high-order derivatives
  - but...problems with the adjoint formulation
- Transformed programs are efficient:
  - Global program analyses are possible and most welcome !
  - The compiler can optimize the generated program.
  - but...some work by the user is needed

How does AD work?

- Program $\mathtt{P}$ is a sequence of instructions $\mathtt{I}_1, \mathtt{I}_2, \ldots, \mathtt{I}_p$
- Each instruction $\mathtt{I}_k$ is a function $\phi_k : \mathbb{R}^{q_{k-1}} \longrightarrow \mathbb{R}^{q_k}$
- The program $\mathtt{P} : \{\mathtt{I}_1, \mathtt{I}_2, \ldots, \mathtt{I}_p\}$ is then

$$x \mapsto y = \Phi(x) = \phi_p \circ \phi_{p-1} \circ \cdots \circ \phi_1(x)$$

- Apply the chain rule

$$\frac{d\Phi}{dx} = \phi_p'(w_{p-1})\phi_{p-1}'(w_{p-2}) \cdots \phi_1'(w_0)$$

where $w_0 = x$ and $w_k = \phi_k(w_{k-1})$

AD: Tangent and Reverse mode

$$\begin{cases} w_0 = x \\ w_k = \phi_k(w_{k-1}) \quad \text{with} \quad k = 1, \ldots, p-1 \\ \\ \dfrac{d\Phi}{dx} = \phi_p'(w_{p-1})\phi_{p-1}'(w_{p-2}) \cdots \phi_1'(w_0) \end{cases}$$

- The full Jacobian $\frac{d\Phi}{dx}$ is expensive (matrix-by-matrix multiplications)
- Use the less expensive matrix-by-vector multiplication
  - Tangent mode: $x, \dot{x} \mapsto \left(\frac{d\Phi}{dx}\right)\dot{x} = \phi_p'(w_{p-1})\phi_{p-1}'(w_{p-2}) \cdots \phi_1'(w_0)\dot{x}$
  - Reverse mode: $x, \bar{\Phi} \mapsto \left(\frac{d\Phi}{dx}\right)^T \bar{\Phi} = \phi_1'^T(w_0) \cdots \phi_{p-1}'^T(w_{p-2})\phi_p'^T(w_{p-1})\bar{\Phi}$
- Theoretical cost is a small multiple of the cost of P [Griewank, 2000]

Example: Tangent differentiation by Program transformation

```
SUBROUTINE FOO  (a,   x,   y   )
    REAL a, x, y



    y = a*y + a

    x = x*y
END
```

Example: Tangent differentiation by Program transformation

```
SUBROUTINE FOO_D(a,ad,x,xd,y,yd)
    REAL a, x, y
    REAL ad, xd, yd

    yd = ad*y + a*yd + ad
    y = a*y + a
    xd = x*yd
    x = x*y
END
```

Example: Tangent differentiation by Program transformation

```
SUBROUTINE FOO_D(a,ad,x,xd,y,yd)
    REAL a, x, y
    REAL ad, xd, yd

    yd = ad*y + a*yd + ad
    y = a*y + a
    xd = x*yd
    x = x*y
END
```

- Tangent AD keeps the structure of the original program `FOO`
- "Differentiated instructions" inserted into `FOO`'s original control flow.

Example: Reverse differentiation

```
SUBROUTINE FOO  (a,   x,   y   )
    REAL a, x, y

    y = a*y + a

    x = x*y



END
```

Example: Reverse differentiation

```
SUBROUTINE FOO_B(a,ab,x,xb,y,yb)
    REAL a, x, y, ab, xb, yb
    CALL PUSHREAL4(y)
    y = a*y + a
    CALL PUSHREAL4(y)
    x = x*y
    CALL POPREAL4(y)
    yb = x*xb
    CALL POPREAL4(y)
    ab = (y+1.0)*yb
    yb = a*yb
    yb = x*xb
END
```

Reverse mode

- Instructions differentiated in the reverse order
- Forward sweep (original code) then backward sweep (differentiation)
- Different approaches to restore the previous values

Reverse mode

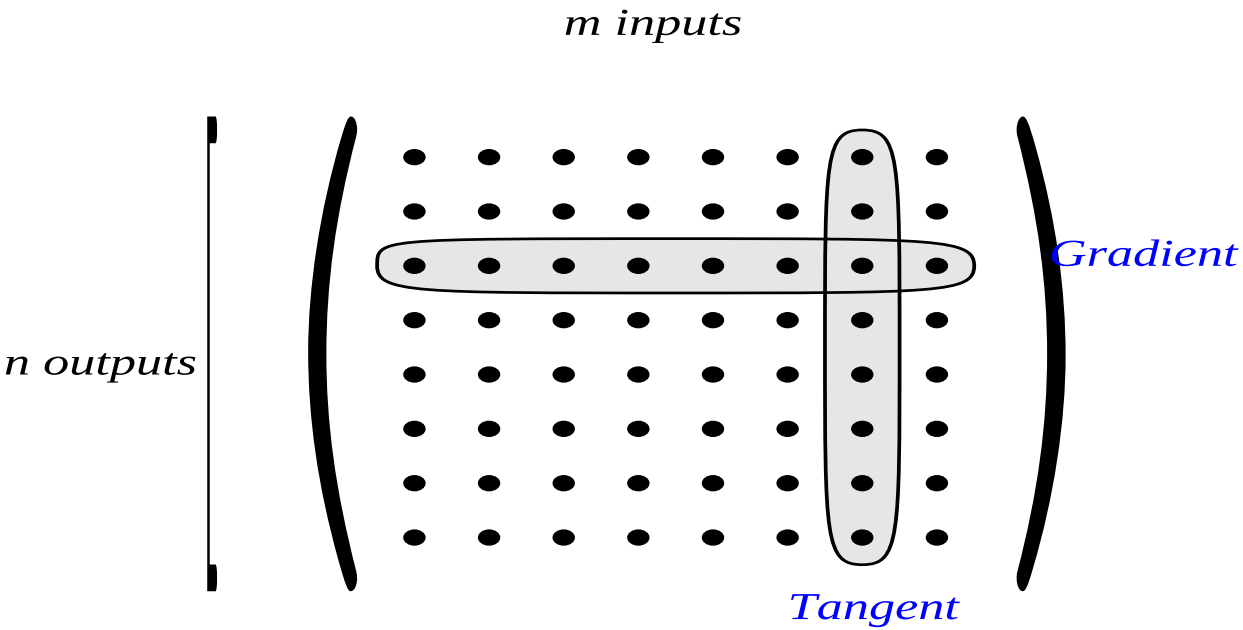
- Instructions differentiated in the reverse order
- Forward sweep (original code) then backward sweep (differentiation)
- Different approaches to restore the previous values
  - Store-All strategy (CPU use: low , memory use: high)
  - Recompute-All strategy (CPU use: high, memory use: low)

Reverse mode

- Instructions differentiated in the reverse order
- Forward sweep (original code) then backward sweep (differentiation)
- Different approaches to restore the previous values
  - Store-All strategy (CPU use: low , memory use: high)
  - Recompute-All strategy (CPU use: high, memory use: low)
- Often a trade-off between SA and RA is needed (Checkpointing)

Why the Reverse mode is useful?

- It gives us the *gradient* of a functional at a cost that is independent from the number of variables



- The Jacobian costs $\alpha_T * m * \mathsf{P}$ with the Tangent mode $(1 < \alpha_T \leq 4)$ [Griewank, 2000] $\Rightarrow$ good if $n > m$
- The Jacobian costs $\alpha_R * n * \mathsf{P}$ with the Reverse mode $(1 < \alpha_R \lesssim 5)$ [Griewank, 2000] $\Rightarrow$ good if $m \gg n$ (for functionals $n = 1$)