Metodo dei minimi quadrati

Caso retta dei minimi quadrati

Partendo dai dati (X,Y) scrivo il sistema delle equazioni normali, nel caso della retta dei minimi quadrati.
Devo trovare la retta $\alpha x + \beta = L(x)$ dove α , β sono le nostre incognite.
Quindi devo trovare i valori di α , β
Creo una matrice formata da tutti 0 (il numero di righe dipende dal numero di condizioni; il numero di colonne dipende dal numero di incognite)
Inizializzo la matrice quindi la matrice dei coefficienti A,
nella prima colonna inserisco i coefficienti del vettore x (sono i coefficienti davanti all'incognitaα)
nella seconda colonna inserisco un vettore formato da tutti 1 (coefficienti davanti all'incognita beta)
Pongo il vettore b uguale al vettore Y , $(Y = L(x))$

In breve ottengo il sistema $A * c = Y$
Dove $c = [\alpha, \beta]^T$ (è il vettore colonna delle incognite

da un punta di visto teorico il sistema risulta sovradimenzionato
quindi dovrei calcolare il sistema di equazioni normali

$$A^T * A * c = A^T * Y$$

In pratica sfrutto il comando \ che in matlab mi da (nel caso di sistemi sovradimensionati) già la soluzione rispetto ai minimi quadrati

Esempio codice

```
A(:,1)=x;

A(:,2)=1;
```

Pongo il vettore b uguale al vettore y

```
b=y;
```

Calcolo i coefficienti della retta dei minimi quadrati

```
alpha_beta=A\b;
```

Trovo un vettore appartenente a R^2 (ho quindi due valori)

Il primo è alpha

```
alpha=alpha_beta(1)
```

il secondo è beta

```
beta=alpha_beta(2)
```

Calcolo il vettore contenente gli errori commessi per ogni singolo punto

```
errore_mq=zeros(7,1);
for i=1:7
    errore_mq(i)=L(x(i))-y(i);
end
errore_mq
```

Interpolazione

L'interpolazione consiste nel trovare un polinomio che "approssima" un'altra funzione o "interpola" dei dati
In pratica dati un numero di punti cerco quel polinomio che passa per quei punti

Su matlab sfruttiamo due comandi

polyfit

p = polyfit(x, y, n)
restituisce i coefficienti di un polinomio p(x) di grado n che costituisce il miglior adattamento (in termini di minimi quadrati) per i dati in y. I coefficienti in p sono in potenze decrescenti e la lunghezza di p è n+1

Osservazione, polyfit ci restituisce i coefficienti del polinomio interpolante in ordine decrescente, cioè

$$p(x) = \text{coeffInt}(1) * x^5 + \text{coeffInt}(2) * x^4 + \text{coeffInt}(3) * x^3 + \text{coeffInt}(4) * x^2 + \text{coeffInt}(5) * x + \text{coeffInt}(6)$$

Nel caso in cui abbiamo n+1 punti (dove n è il grado del polinomio) polyfit troverà i coefficienti del polinomio interpolante (il polinomio che passa per quei punti)

Nel caso in cui abbiamo più di n+1 punti, ci ritroviamo di fronte a un problema sovradimensionato, e il comando polyfit restituirà i coefficienti del polinomio p(x) di grado n che restituisce il miglior adattamento (in termini dei minimi quadrati) per i dati y.

polyval

y = polyval(p,x)
valuta il polinomio p in ciascun punto di x. L'argomento p è un vettore di lunghezza n+1 i cu elementi sono i coefficienti (in potenze decrescenti) di un polinomio di grado n-esimo.

Quindi
p = polyfit(x, y, n)
y = polyval(p, x)

p sarà quindi un vettore contenente le valutazioni (le ordinate del polinomio interpolante calcolate) dei punti x

Esempio codice

```
y=zeros(4,1);
for i=1:4
    y(i)=f(x(i));
end


coeffXY=zeros(4,2);
coeffXY(:,1)=x;
coeffXY(:,2)=y;
```

```
p=polyfit(x,y,3);
polInt=polyval(p,z);
```

nel caso in cui devo calcolare l'ordinata in un punto specifico del polinomio interpolante mi basterà usare il comando polyval nel punto specifico

```
x4=0.8;
xInt= polyval(p,x4)
```

Interpolazione caso curve parametriche

Nel caso in cui io voglia trovare la parametrizzazione di una curva, dovrò trovare le funzioni parametriche,
cioè due polinomi che gestiscano la x e la y

$$p_x(i) = X_i, p_y(i) = Y_i, i \in \{0, 1, 2, 3, 4, 5\}.$$

```
indice = (0:5);
X = [0, 0, -2, 0, 4, 0];
Y = [0, 1, 0, -3, 0, 5];


x = linspace(0,5);
y = linspace(0,5);


coeffpx = polyfit(indice, X, 5);
coeffpy = polyfit(indice, Y, 5);
px = polyval(coeffpx, x);
py = polyval(coeffpy, y);


plot(px, py, "k-")
```

Interpolazione Chebyshev

Ricordo che il polinomio di Chebyshev di grado n è

$T_n(x) = \cos(n * \varphi)$ dove $x = \cos(\varphi)$

i nodi di Chebyshev sono gli zeri di tale polinomio

quindi $\cos(n * \varphi) = 0$

cioè

$n * \varphi_k = \frac{\pi}{2} + k * \pi$ dove $k = 0, 1, \ldots, n-1$

$\varphi_k = \frac{\pi}{2 * n} + \frac{k}{n} * \pi$ dove $k = 0, 1, \ldots, n-1$

in questo modo trovo n nodi

Però per trovare un polinomio interpolante di grado n ho bisogno di $n+1$ nodi,

mi basterà quindi cercare gli zeri del polinomio di Chebyshev

$T_{n+1}(x) = \cos((n+1) * \varphi)$ dove $x = \cos(\varphi)$

quindi

$\varphi_k = \frac{\pi}{2 * (n+1)} + \frac{k}{n+1} * \pi$ dove $k = 0, 1, \ldots, n$

chiamo $x_k = \cos\left(\frac{\pi}{2 * (n+1)} + \frac{k}{n+1} * \pi\right)$ dove $k = 0, 1, \ldots, n$

in modo che $T_n(x_k) = 0$ per $k = 0, 1, \ldots, n$

gli x_k sono proprio i nodi cercati

Esempio n = 5

dobbiamo cercare un polinomio $p_5(x)$ di interpolazione di grado 5

quindi $n = 5$

sostituisco o ottengo

$x_k = \cos\left(\frac{\pi}{2 * 6} + \frac{k}{6} * \pi\right)$ dove $k = 0, 1, \ldots, 5$

$x_k = \cos\left(\frac{\pi}{12} + \frac{k}{6} * \pi\right)$ dove $k = 0, 1, \ldots, 5$

calcolo le φ_k e dopo le x_k

```
angoliPh = zeros(6,1);
for i=1:6
    angoliPh(i) = pi/12 + pi/6 * (i-1);
end
angolicercati = angoliPh


nodiC = zeros(6,1);
for i=1:6
    nodiC(i) = cos(angoliPh(i));
end
nodicercati = nodiC
```

calcolo le ordinate dei nodi per sfruttare i comandi polyfit e polyval, per poi calcolare trovare il polinomio

```
x = -1:0.1:1;
f = @(x) exp(2*x);


ordinateN = zeros(6,1);
for i=1:6
    ordinateN(i) = f(nodiC(i));
end
ordinateNodi = ordinateN


coeffInt = polyfit(nodiC, ordinateN, 5)
polInt = polyval(coeffInt, x);
```

Metodo di Newton

Dal punto di vista teorico scelgo $[a_1, b_1]$

ricordo che le condizioni di convergenza sono

- $F(a) * F(b) < 0$
- La funzione $F(x)$ sempre convessa (o concava) in tutto l'intervallo $[a, b]$

In pratica scelgo un punto di partenza x_0 abbastanza vicino alla soluzione esatta, in modo da assicurare la convergenza del metodo

Script

Metodo di Newton

```
function xnew = newton (xold, toll, iterMax)
    a = 1.5;
    b = 1.5;
    c = 1;
    F = @(x) x.^2/a^2 + c*x.*(cosh(x)-2) + (cosh(x)).^2/b^2 -1 ;
    dPrimaF = @(x) 2*x/a^2 + sinh(2*x)/b^2 + c*cosh(x) + c*x.*sinh(x) - 2*c ;
    iter = 0;
    err = 10;
    while err > toll && iter < iterMax
        xnew = xold - (F(xold)/dPrimaF(xold));
        err = abs(xnew - xold);
        xold = xnew;
        iter = iter +1;
    end
end
```

Possiamo usare il metodo di Newton anche per calcolare un punto di intersezione fra due curve

$$\begin{cases} f(x) \\ g(x) \end{cases}$$

Quadratura e ordine polinomiale

L'ordine polinomiale di una formula di quadratura è

$m \leq 2 * n + 1$

(n rappresenta il $\text{numero di nodi} - 1$)

per determinare l'ordine polinomiale di una formula di quadratura mi basta verificare che

$I_n[x^i] = I[x^i]$ per $i = 0, 1, \ldots, 2 * n + 1$ (al massimo m = 2*n +1)

e t.c.

$I_1[x^{k+1}] \neq I[x^{k+1}]$

l'ordine polinomiale sarà quindi $m = k$

Teorema utile

se $F(x)$ è una funzione dispari, cioè $F(-x) = -F(x)$ allora

$$\int_{-a}^{a} F(x)\ \mathrm{dx} = 0$$

Def Sia m un intero non negativo. Una formula di quadratura I_n a n + 1 nodi si dice di ordine polinomiale m se la formula di quadratura fornisce il valore esatto dell'integrale quando è applicata ad un qualsiasi polinomio di grado ≤ m.

Quadratura Gaussiana

la generica formula Gaussiana è del tipo la generica formula Gaussiana è del tipo

$I_n[f] = \sum_{i=0}^{n} w_i * f(x_i)$

per determinare una formula Gaussiana devo calcolare i nodi w_i e i pesi x_i

Inoltre una formula Gaussiana ha ordine polinomiale massimo, cioè

$m = 2 * n + 1$

Quindi posso imporre le condizioni da

$I_n[x^i] = I[x^i]$ per $i = 0, 1, \ldots, 2 * n + 1$

Quadratura Gauss-Chebyshev

N numero di intervalli

N nodi polinomio di Chebyshev

la formula di quadratura di Gauss Chebyshev è del tipo

$$\mathrm{GC} = \int_{-a}^{a} w(x) * f(x) = \sum_{i=1}^{N} w_i * f(x_i)$$

nel caso di Gauss Chebyshev

$$w(x) = \frac{1}{\sqrt{1 - x^2}}$$

ed x_i sono i nodi del polinomio di Chebyshev $T_N = \cos(n * \phi)$ dove $x_i = \cos(\phi)$ con $\phi \in [0, \pi]$

$n * \phi = \frac{\pi}{2} - k * \pi$ dove $k = 0, 1, .. N - 1$

$\phi = \frac{\pi}{2 * n} - k * \frac{\pi}{n} = \frac{\pi}{n} * \left(\frac{1}{2} - k\right)$ dove $k = 0, 1, .. N - 1$

quindi devo riuscire a riscrivere la funzione come

$$F(x) = \frac{1}{\sqrt{1 - x^2}} * f(x)$$

inoltre nel caso di Gauss Chebyshev ho che

$$w_i = \frac{\pi}{N}$$

cioè la grandezza dell'iintervallino ottenuto dividendo $[0, \pi]$ in N intervallini

quindi

$$\mathrm{GC} = \sum_{i=1}^{N} \frac{\pi}{N} * f(\cos(\phi_i))$$

Esempio script

Script per solo 16 nodi

```
F = @(z) 2/( sqrt(2*(1+z.^2)) );
N1 = 16;
wj = pi/N1;


nodiC = zeros(N1);
GN1 = 0;
for i = 1 : N1
    nodiC(i) = cos((1/2 - i) * wj );
    GN1 = GN1 + wj * F(nodiC(i));
end
GN1
```

Cholesky

Sia data S matrice quadrata, la fattorizzazione di Cholesky consiste nel trovare una matrice R t.c.

S = R' * R (dove R' = trasposta di R)

le condizioni per fattorizzare tramite Cholesky la matrice sono :

- S simmetrica (cioè S^T=S)
- S definita positiva

Per dimostrare che S sia definita positiva posso sfruttare un corollario:

- Una matrice simmetrica, a diagonale dominante, i cui elementi diagonali sono tutti >0, è definita positiva

Codice

```
R=chol(S)
```

Restituisce la matrice della fattorizzazione di Cholesky

Metodo delle potenze

Il metodo delle potenze permette di calcolare l'autovalore di modulo massimo di un matrice A quadrata

Condizioni di convergenza:

- Esiste unico autovalore di modulo massimo (per forza di cose è un autovalore reale)

il metodo delle potenze consiste nel creare una successione di vettori $\{x^{(0)}, x^{(1)}, \ldots, x^{\dot{x}}, \ldots\}$

in questo modo

$x^{(k+1)} = A * x^{(k)}$

sotto le opportune condizioni $x^{(k)} \to u$ autovettore

mentre il valore

$\sigma_k = \frac{x^{(k)\,T} * A * x^{(k)}}{x^{(k)\,T} * x^{(k)}}$ (quoziente di Rayleigh)

con $\sigma_k \to \lambda_1$ (converge all'autovalore di modulo massimo

osservo che l'autovettore u è l'autovettore associato all'autovalore λ_1

Per evitare gli errori di overflow e underflow normalizzo i vettori x_k

riscrivo quindi il metodo

$$\begin{cases} y^{(k)} = \frac{x^{(k)}}{\|x^{(k)}\|} \\ x^{(k+1)} = A * y^{(k)} \end{cases}$$

$$\sigma_k = \frac{x^{(k)\,T} * A * x^{(k)}}{x^{(k)\,T} * x^{(k)}} = \frac{y^{(k)\,T} * A * y^{(k)}}{y^{(k)\,T} * y^{(k)}}$$

$$y^{(k)} \longrightarrow \frac{u}{\|u\|}$$

$\sigma_k \longrightarrow \lambda_1$

Un possibile criterio di arresto è $\|A * x^{(k)} - \sigma_k * x^{(k)}\| < \text{toll} * \|x^{(k)}\|$ (sfrutto il "residuo")

Script

```
x0 = [1; 1; 1; 1; 1];
xk = x0;
y = xk/norm(xk);
iterEs = 40;
for i=1:iterEs
    xk = A * y;
    y = xk/norm(xk);
end
sigmaRay = y' * A * y /(y' * y);
disp(["autovalore di modulo massimo ", sigmaRay])
disp(["autovettore associato ", xk']) %ho messo trasposto perchè lo vuole riga
disp(["autovettore normalizzato", y'])
```

Metodo delle potenze inverse

Il metodo delle potenze inverse permette invece di trovale l'autovalore di modulo minimo

sfruttando il metodo delle potenze alla matrice inversa A^{-1} mi calcolo l'autovalore di modulo massimo di tale matrice inversa,

poi sfruttando un corollario che afferma che

data una matrice A invertibile, e chiamati λ_k gli autovalori della matrice A

ho che gli autovalori α_k della matrice inversa A^{-1}, saranno $\alpha_k = \frac{1}{\lambda_k}$

quindi per forza di cose l'autovalore di modulo minimo della matrice inversa A^{-1} sarà $\alpha_n = \frac{1}{\lambda_1}$ (λ_1 autovalore di modulo massimo della matrice A)

mentre l'autovalore di modulo massimo della matrice inversa A^{-1} sarà $\alpha_1 = \frac{1}{\lambda_n}$ (λ_n autovalore di modulo minimo della matrice A)

applico quindi il metodo delle potenze alla matrice inversa A^{-1} calcolandone l'autovalore α_1 di modulo massimo

calcolo dunque l'autovalore $\lambda_n = \frac{1}{\alpha_1}$ che sarà quindi autovalore di modulo minimo della matrice A

$$x^{(k+1)} = A^{-1} * x^{(k)}$$

in quanto calcolare la matrice inversa A^{-1} è oneroso in termini di costo computazionale preferisco calcolare un sistema lineare riscrivendo il metedo come

$$A * x^{(k+1)} = x^{(k)}$$

dove $x^{(k+1)}$ è la nostra incognita (uso il comando $A \setminus x^{(k)}$ per calcolare $x^{(k+1)}$)

normalizzo per evitare errori di overflow e underflow

$$y^{(k)} = \frac{x^{(k)}}{||x^{(k)}||}$$

$$x^{(k+1)} = A^{-1} * y^{(k)}$$

ottengo quindi

$$A * x^{(k+1)} = y^{(k)}$$

il quozionte di Rayleigh sarà

$$\sigma_k = \frac{x^{(k)\,T} * A^{-1} * x^{(k)}}{x^{(k)\,T} * x^{(k)}} = \frac{y^{(k)\,T} * A^{-1} * y^{(k)}}{y^{(k)\,T} * y^{(k)}}$$

riscrivo

$$(y^{(k)\,T} * y^{(k)}) * \sigma_k = y^{(k)\,T} * A^{-1} * y^{(k)}$$

ma $x^{(k+1)} = A^{-1} * y^{(k)}$ quindi

$(y^{(k)\,T} * y^{(k)}) * \sigma_k = y^{(k)\,T} * x^{(k+1)}$

sfruttondo i comandi matlab posso calcolare $x^{(k+1)}$ con $A \setminus y^{(k)}$

e a sua volta da $(y^{(k)\,T} * y^{(k)}) * \sigma_k = y^{(k)\,T} * x^{(k+1)}$ calcolo il quazione di Rayleigh come $(y^{(k)\,T} * y^{(k)}) \setminus y^{(k)\,T} * x^{(k+1)}$

sostituendo tutto ottengo

$\sigma_k = (y^{(k)\,T} * y^{(k)}) \setminus \left(y^{(k)\,T} * (A \setminus y^{(k)}) \right)$

il procedimento per matlab sarà

$$y^{(k)} = \frac{x^{(k)}}{\|x^{(k)}\|}$$

$x^{(k+1)} = A \setminus y^{(k)}$

e

$\sigma_k = (y^{(k)\,T} * y^{(k)}) \setminus \left(y^{(k)\,T} * (A \setminus y^{(k)}) \right)$

Script

```
xInvk = x0;
yInv = xInvk/norm(xInvk);


for i=1:iterEs
    xInvk = A \ yInv;
    yInv = xInvk / norm(xInvk);
end
sigmaRayInv = (yInv' * yInv) \ (yInv' * (A\yInv));
autovalMin = 1/sigmaRayInv;
disp(["autovalore di modulo minimo ", autovalMin])
disp(["autovettore associato ", xInvk'])
disp(["autovettore normalizzato", yInv'])
```

Metodo delle potenze inverse (caso ricerca del raffinamento di un’approssimazione di autovettore)

l'idea è di avvicinarci all'autovalore cercato patrendo da un’approssimazione

Dalla teoria:

sia z_0 approssimazione dell'autovalore desiderato,

calcolo il metodo delle potenze inverse alla matrice $A_z = A - z_o\, I$

così otterremo l'autovalore minimo per la matrice A_z

a quel punto mi basta osservare che tale autovalore sarà $\lambda_z = \lambda_A - z_0$

quindi $\lambda_A = \lambda_z + z_0$

script (autovalore più vicino a 3)

```
A3 = [-1  2  5
      1  -3  1
      1 -2  -1];


x0 = [1; 1; 1];
ynew = x0/norm(x0);
for i= 1 : 10
    xnew = A3 \ ynew;
    ynew = xnew / norm(xnew);
end


reil = (ynew'* ynew) \ (ynew' * (A3 \ ynew))
autovMin = 1/reil
autovEs = autovMin + 3
```

Teorema di Geshgorin

il primo teorema di Gershgorin afferma che

data una matrice $A \in C^{n*n}$, siano λ gli autovalori della matrice A

gli autovalori $\lambda_k \in \Omega = \underset{k}{U} D_k$

dove $D_k = \{z \in C : |z - a_{kk}| < r_k\}$ (dischi)

con $r_k = \sum_{j=1 \quad j \neq k}^{n} a_{kj}$ (raggi)

script

```
format long
n = length(A);
C=zeros(1,n);
r=zeros(1,n);
for k = 1:n
    C(k) = A(k,k);
    r(k) = sum(abs(A(k,:))) - abs(A(k,k));
end
disp(["I centri sono :", C, "I raggi sono :", r])


minimo = min(C-r);
massimo = max(C+r);


disp(['Gli autovalori della matrice A sono compresi nell''intervallo
[',num2str(minimo), ' , ' num2str(massimo),']' ])
```

Script funzione cerchio

scrivo la funzione cerchio

ricordando le formule parametriche della circonferenza

$$\begin{cases} x = x_0 + \rho * \cos(\theta) \\ y = y_0 + \rho * \sin(\theta) \end{cases}$$

```
function cerchio(x0,y0,r)
    theta = linspace(0, 2*pi, 360);
    x = x0 + r * cos(theta);
    y = y0 + r * sin(theta);
    patch(x, y, "g")
end
```

Stampare i cerchi

```
for k = 1:n
    cerchio(C(k),0,r(k))
    hold on
end
grid on
axis equal
hold off
```

Comando eig

uso il comando eig per trovare gli autovalori

```
autovalori = eig(A)
autovaloreMin = min(abs(autovalori))
autovaloreMax = max(abs(autovalori))
```

Quadratura composita

Trapezi

metodo dei trapezi per approssimare l'integrale

$$T_N = \sum_{i=0}^{N-1} (f(x_i) + f(x_{1+1})) * \frac{(x_{i+1} - x_i)}{2}$$

oppure

posto $h = \frac{b-a}{N}$

$T_N = \left(f(x_0) + 2 * \sum_{i=1}^{N-1} f(x_i) + f(x_N) \right) * \frac{h}{2}$ dove $x_0 = a$ e $x_N = b$

Script

```
a = 0;
b = 1;
N = 10;                                          % Numero di intervalli
nodi = linspace(a, b, N + 1);                    % I nodi sono (numero di intervalli)
+ 1


T10 = 0;                                         % Inizializzo la variabile per il
metodo dei trapezi
for i = 1:N
    T10 = T10 + (F(nodi(i)) + F(nodi(i+1))) * (nodi(i+1) - nodi(i)) / 2;
end
T10


h = (b-a)/N;
T10sum = (F(nodi(1)) + 2 * sum(F(nodi(2:N))) + F(nodi(N+1))) * h/2           %
Altro modo per calcolare il metodo dei trapezi
```

Simpson

$$S_N = \sum_{i=0}^{N-1} \left(F(x_i) + 4 * F\left(\frac{x_i + x_{i+1}}{2}\right) + F(x_{i+1}) \right) * \frac{(x_{i+1} - x_i)}{2} * \frac{1}{3}$$

osservo che $\frac{x_i + x_{i+1}}{2}$ è il punto medio tra x_i e x_{i+1}

sviluppo i prodotti

$$S_N = \frac{1}{6} \sum_{i=0}^{N-1} \left(F(x_i) + 4 * F\left(\frac{x_i + x_{i+1}}{2}\right) + F(x_{i+1}) \right) * (x_{i+1} - x_i)$$

se gli N intervalli sono tutti uguali tra loro

posso porre $h = \frac{(x_{i+1} - x_i)}{2}$

ottendendo la forma compatta (tenendo conto che $\frac{h}{6} = \frac{(x_{i+1} - x_i)}{2} * \frac{1}{3}$)

$$S_N = \frac{h}{3} * \sum_{i=0}^{N-1} \left(F(x_i) + 4 * F\left(\frac{x_i + x_{i+1}}{2}\right) + F(x_{i+1}) \right)$$

noto anche che

$$x_{i+1} - x_i = \frac{b - a}{N}$$

quindi $h = \frac{(x_{i+1} - x_i)}{2} = \frac{b - a}{N} * \frac{1}{2} = \frac{b - a}{2 * N}$

Script

```
H = (b-a)/(2*N);


S10 = 0;
for i = 1:N
    S10 = S10 + ( F(nodi(i)) + 4* F((nodi(i)+nodi(i+1))/2) + F(nodi(i+1))  );
end
S10 = H/3  *  S10
```

Punto Medio

$$\mathrm{PM}_N = \sum_{i=0}^{N-1} F\left(\frac{x_i + x_{i+1}}{2}\right) * (x_{i+1} - x_i)$$

se gli N intervalli sono tutti uguali tra loro

$$x_{i+1} - x_i = \frac{b-a}{N} = h$$

$$\mathrm{PM}_N = h * \sum_{i=0}^{N-1} F\left(\frac{x_i + x_{i+1}}{2}\right)$$

Script

```
hPM = (b-a)/N;


PM = 0;
for i = 1:N
    PM = PM + F( (nodi(i) + nodi(i+1) )/2   );
end
PM = hPM * PM
```

Numero di condizionamento

Data un sistema lineare $A * x = b$

e suppongo di perturbare i dati $A * (x + \Delta x) = b + \Delta b$

mi chiedo in che modo queste perturbazioni influiranno sul risultato finale,

cioè quanto sarà l'errore provocato da tali perturbazioni.

Dirò inoltre che un problema è **ben condizionato** quando la soluzione del problema perturbato non differirà molto dalla soluzione del problema originale

al contrario un problema si dirà **mal condizionato** quando la soluzione del problema perturbato sarà molto diversa dalla soluzione del problema originale

chiamo **numero di condizionamento** il valore $\mu(A) = \|A\| * \|A^{-1}\|$

il numero di condizionamento dipende dalla matrice e dalla norma utilizzata per calcolarlo

Il numero condizionamento è quindi un valore $\mu(A) \geq 1$

Quando il numero di condizionamento è vicino a 1 il problema sarà **ben condizionato**, quando invece è molto più grande di 1 il problema sarà **mal condizionato.**

Script calcolo numero di condizionamento

```
      HInv = inv(H);


    c1(n) = norm(H,1) * norm(HInv, 1);
    c2(n) = norm(H,2) * norm(HInv, 2);
    cInf(n) = norm(H, Inf) * norm(HInv, Inf);
```

Comando matlab

```
cCond(n) = cond(H);
```

cond(H) restituisce il numero di condizionamento calcolato su norma 2

contour

```
format long
x = -2:0.02:4;
y = -2:0.02:5;
circonferenza = @(x,y) x.^2 + y.^2 - 2*x - 4*y + 1;


f = @(x) exp(x);


[X,Y] = meshgrid(x,y);
contour(X, Y, circonferenza(X,Y), [0, 0], "k-")
grid on
axis equal
```

Il comando contour (X, Y, F(x,y), [0,0])
Permette di plottare una curva

Osservazione
Faccio in modo di scrivere F(x,y) = 0
Quindi porto tutto a primo membro

E uso [0,0] come linee di livello per trovare la curva desiderata

Trasformazioni affini

$x = \alpha * t + \beta$
Dove $x \in [a, b]$ $t \in [c, d]$

Pongo

$$\begin{cases} a = \alpha * c + \beta \\ b = \alpha * d + \beta \end{cases}$$

Devo trovare α, β

Esempio

Passo da $x \in [-1,1]$ a $t \in [-\pi, \pi]$

$$\begin{cases} -1 = -\pi * \alpha + \beta \\ 1 = \pi * \alpha + \beta \end{cases}$$

Sommo
$0 = 2 * \beta$ $\beta = 0$

Sottraggo
$2 = 2 * \pi * \alpha$ $\alpha = \frac{1}{\pi}$

Quindi la trasformazione affine è
$x = \frac{1}{\pi} * t$ quindi se conosco x allora $t = \pi * x$

Trasformazione affina (quadratura)

$\int_a^b f(x)dx$ devo passare dall'intervallo $x \in [a,b]$ a $t \in [c,d]$

$$x = \alpha * t + \beta$$
$$dx = \alpha * dt$$

$$\int_a^b f(x)dx = \int_c^d f(\alpha * t + \beta) * \alpha * dt$$

Esempio

Passo da $x \in [-1,1]$ a $t \in [-\pi,\pi]$

Per i passaggi precedenti

$$x = \frac{1}{\pi} * t$$
$$dx = \frac{1}{\pi} * dt$$

$$\int_{-1}^{1} f(x)dx = \int_{-\pi}^{\pi} f\left(\frac{1}{\pi} * t\right) * \frac{1}{\pi} * dt$$

Sviluppo in serie di Taylor

$$\sum_{i=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} * (x - x_0)^n$$

Funzioni iperboliche

$$\sinh(x) = \frac{e^x - e^{-x}}{2}$$

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

Derivate

$\sinh'(x) = \cosh(x)$

$\cosh'(x) = \sinh(x)$

Funzioni utili

$f(x) = \ln(x)$

$f'(x) = \frac{1}{x}$

$\ln(a * b) = \ln(a) + \ln(b)$