ese_extra_parte1_traccia

March 20, 2024

1 Esercizi extra - parte 1

In questo notebook troviamo esercizi relativi alla prima parte del programma di numerica, di vario grado di difficoltà (indicato nel testo).

2 Zeri di funzioni

2.0.1 Esercizio 1

Difficoltà: facile. Secondo l'equazione dei gas perfetti il volume specifico v dell'aria è dato da

$$v = \frac{TR^\star}{p}$$

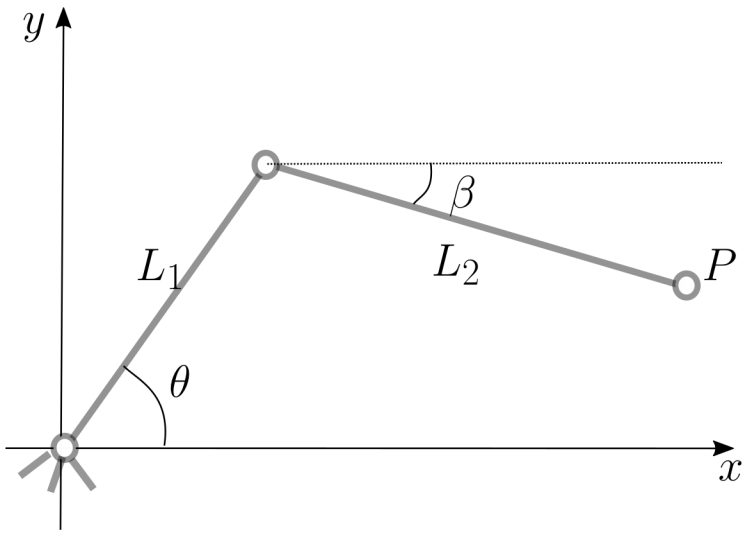
dove $R^\star = 287.78\, J\, kg^{-1}\, K^{-1}$. Per $T = 288\, K$, $p = 101325\, Pa$ si ottiene $v = 0.8180$ ossia densitá $\rho = 1.225\, kgm^{-3}$. Consideriamo ora la legge dei gas reali

$$f(v) = \left(p + \frac{a}{v^2}\right)(vb - c) - dR^\star T = 0.$$

con $a = 1.2425e03$, $b = 2.8890e-05$, $c = 3.8700e-05$, $d = 1.0e-5$ (dati non realistici). Risolvere l'equazione dei gas reali con il metodo di bisezione partendo da un intervallo iniziale adeguato. Che densità si ottiene? Calcolare il risultato con un errore massimo di $\epsilon = 1.0e-5$ stabilendo a priori il numero di iterazioni necessarie dato l'intervallo scelto.

2.0.2 Esercizio 2

Difficoltà: difficile. Si consideri il cinematismo in figura e si descriva la posizione del punto finale P utilizzando i due angoli e il sistema di riferimento indicati.



Imponendo $P = (2.5, 0)$, $L_1 = 2$, $L_2 = 1$, che angoli θ e β si ottengono? Risolvere il sistema non-lineare con il metodo di Newton per sistemi per diversi valori della guess iniziale, con tolleranza $\epsilon = 10^{-6}$. La soluzione è unica? Abbiamo sempre convergenza? Perché?

2.1 Sistemi lineari

2.1.1 Esercizio 3

Difficoltà: media.

Si costruisca la matrice di Vandermonde di dimensione 10×10 con il comando np.vander. L'elemento vij di tale matrice è costruito come $v_{ij} = x_i^{n-j-1}$ il numero di righe e xi sono gli elementi del vettore dato in input. La matrice di Vandermonde può essere usata nella costruzione dei polinomi interpolanti su nodi x_i ed è notoriamente malcondizionata.

[46]:
```
V=np.vander(np.linspace(1,10,10))
```

Si costruisca un termine noto $\mathbf{b}$ tale che la soluzione esatta sia $x_e = (1, 1, 1, 1, \dots, 1)^T$.

Si costruisca ora un termine noto perturbato $\mathbf{b}+\delta\mathbf{b}$ dove la perturbazione $\delta\mathbf{b}$ è un vettore di numeri random compresi fra 0 e 0.1. Suggerimento: usare il comando numpy.random.rand(nrows,ncols) e riscalare il vettore ottenuto.

Si risolva il sistema perturbato con la fattorizzazione LU, verificando se è stato eseguito il pivoting. Si calcoli la soluzione $\mathbf{x} + \delta\mathbf{x}$ utilizzando i metodi di sostituzione opportuni.

Si verifichi se è soddisfatta la stima teorica relativa alla stabilità del metodo di eliminazione di Gauss che prevede che

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq K(V)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}$$

dove K(V) è il condizionamento della matrice e $\delta\mathbf{x}$ si ottiene per differenza con la soluzione esatta (nota).

2.1.2 Esercizio 4

Difficoltà: dfficile.

Si consideri la matrice A allegata, da importare con il comando

[53]:
```
matrix = np.loadtxt('matrice.txt')
A=scipy.sparse.coo_matrix((matrix[:,2], (matrix[:,0].T.astype(int),matrix[:,1].
  ↪T.astype(int)))).toarray()
```

Si costruisca il termine noto b del sistema lineare $A\mathbf{x} = \mathbf{b}$ tale che la soluzione esatta sia $\mathbf{x}_e = [1, 1, ...]^T$. (Nota: determinare le dimensioni della matrice A con il comando shape.

Calcolare il numero di condizionamento della matrice.

Vogliamo risolvere il sistema lineare con il metodo di Richardson statico. Calcolare il parametro α_{max} massimo per avere convergenza, e il parametro α_{opt} che garantisce il raggio spettrale minimo per la matrice di convergenza considerando come precondizionatore la matrice identità.

Implementare una function per risolvere il sistema lineare con il metodo di Richardson. La funzione riceve in input la matrice, il termine noto, la guess iniziale x_0 , il precondizionatore P, il parametro α, la tolleranza tol e il numero massimo di iterazioni nmax, e restituisce in output il vettore soluzione e il numero di iterazioni effettuate.

Risolvere il sistema con la function scritta e i seguenti dati: il vettore nullo come guess iniziale, precondizionatore pari alla matrice identità, $\alpha = \alpha_{opt}$, nmax=1000 e tol=1.0e-6. Cosa si osserva?

Eseguire la fattorizzazione LU della matrice A e ripetere il punto precedente usando come precondizionatore P = L. Cosa si osserva? Come si spiega questo risultato? (Calcolare il numero di condizionamento di $L^{-1}A$).

2.1.3 Esercizio 5

Difficoltà: media.

Si consideri la matrice A allegata, da importare con il comando

[62]:
```
matrix = np.loadtxt('matrice.txt')
A=scipy.sparse.coo_matrix((matrix[:,2], (matrix[:,0].T.astype(int),matrix[:,1].
  ↪T.astype(int)))).toarray()
```

Si costruisca il termine noto **b** del sistema lineare $A\mathbf{x} = \mathbf{b}$ tale che la soluzione esatta sia $\mathbf{x}_e = [1, 1, ...]^T$. (Nota: determinare le dimensioni della matrice A con il comando shape).

Calcolare il numero di condizionamento della matrice.

Verificare che la matrice è simmetrica e definita positiva.

Risolvere il sistema con il metodo del gradiente (gdescent) e i seguenti dati: il vettore nullo come guess iniziale, nmax=1000 e tol=1.0e-6. Cosa si osserva?

Notiamo che la soluzione è molto lontana da quella esatta, la convergenza è estremamente lenta.

Verificare la stima teorica per l'abbattimento dell'errore con il metodo del gradiente.

Ora, risolvere lo stesso problema, con gli stessi dati, utilizzando il metodo del gradiente coniugato allegato di seguito. Il metodo raggiunge la tolleranza desiderata? In quante iterazioni? Verificare che la convergenza soddisfa la stima teorica.

[85]:
```
def cgdescent(A, b, x0, nmax = 1000, rtoll = 1e-6):
  """
  Metodo del gradiente a parametro dinamico per sistemi lineari.

  Input:
   A      Matrice del sistema
   b      Termine noto (vettore)
   x0     Guess iniziale (vettore)
   nmax   Numero massimo di iterazioni
   toll   Tolleranza sul test d'arresto (sul residuo relativo)

  Output:
   xiter  Lista delle iterate

  """
  norm = np.linalg.norm

  bnorm = norm(b)

  r = b - A@x0

  xiter = [x0]
  iter = 0
  z = r

  while((norm(r) / bnorm)>rtoll  and iter < nmax):
      xold = xiter[-1]

      rho = np.dot(r, z)
      q = A @ z;
      alpha = rho / np.dot(z, q)
      xnew = xold + alpha * z
      r = r - alpha*q
      beta = (r.T @ A @ z)/(z.T @ A @ z)
      z = r - beta*z
      xiter.append(xnew)
      iter = iter + 1

  return xiter
```