Studio per lo sviluppo di Modelli Idrodinamici riguardanti gli aspetti di Interazione Onda-Corrente

Ing. Alessandra Romolo

Tutor: Ing. Francesco Lalli

1. Introduzione

Molteplici sono le azioni che contribuiscono a determinare la morfologia della fascia costiera, intesa come quella porzione di territorio comprendente tra la linea di battigia e la zona di frangimento (in corrispondenza della quale l'onda si rompe e subisce una variazione del suo moto da oscillatorio a traslatorio).
Le dinamiche che entrano in gioco risultano particolarmente complesse, in quanto derivano da diversi fenomeni fisici interagenti tra loro e concorrenti nella formazione della configurazione complessiva. L'innesco di tali fenomeni è dovuto all'azione del moto ondoso, che dal largo si dirige verso la costa, e produce azioni dirette sia sul litorale che su tutte le strutture poste a protezione della fascia costiera.
Tali azioni possono anche subire alterazioni e distorsioni indotte, sia da correnti, che dall'interazione con i fiumi, in corrispondenza delle foci. Le dinamiche che si innestano risultano particolarmente complesse e di notevole interesse, in quanto vengono determinate alterazioni, sia nelle sollecitazioni indotte su eventuali strutture, sia nell'evoluzione e nelle dinamiche generatrici della morfologia costiera.

Oggetto della presente tesi sarà lo studio di tali processi, che riguardano la propagazione di onde di mare in presenza di correnti (ad esempio, in presenza di foci di fiumi), e la loro possibile interazione con strutture costiere dalla generica geometria.

Nelle applicazioni pratiche dell'ingegneria costiera, caratterizzate da fenomeni su ampia scala nel dominio spazio/tempo, queste complesse interazioni sono spesso affrontate nell'approccio classico del *radiation-stress* proposto da Longuet-Higgins e Stewart (1964), in cui vengono considerate le componenti del flusso stazionario del moto ondoso.
La propagazione del moto ondoso nelle regioni costiere è ampiamente studiata anche con i metodi denominati "*mildslope*" (Berkhoff, 1972; Ito e Tanimoto, 1972; Gao e Radder, 1998).
Per l'analisi e la previsione della diffrazione del moto ondoso prodotto da una secca (*shoal*) e per l'interazione di onde con correnti, è stato più recentemente proposto un modello per flusso potenziale (Tenget ed altri (2001), interamente 3-D), che risulta esatto al primo ordine di approssimazione, dal momento che la funzione di Green - nella formulazione integrale - soddisfa le condizioni lineari al contorno della superficie libera. Questo modello limita, però, l'interazione *onde - correnti* al solo caso di flusso stazionario, anche se molte configurazioni implicano un flusso viscoso non uniforme, come nel caso di interazione onda-getto.
Un approccio 3-D totalmente non lineare, non stazionario, per un flusso potenziale a superficie libera, è stato proposto da Lalli et al. (1996) e da Lalli (1997).

Le predette esperienze costituiscono la base su cui è stato approfondito – nel presente studio - l'aspetto connesso all'interazione tra il flusso di un fiume immissario con il moto ondoso, con le correnti indotte e con le strutture costiere; una configurazione che è la risultante della combinazione di due campi di moto riconoscibili nell'ambito della meccanica dei fluidi, noti come "*jet in a cross-flow*" and the "*impact jet on a wall*".

E' stata trascurata la presenza di flusso *long-shore* , considerando soltanto il termine di forzante dovuto al deflusso fluviale; ne consegue che sono stati assunte come più importanti le interazioni tra il flusso proveniente dal fiume e il moto ondoso e con le strutture portuali.

L'analisi numerica è stata condotta con impiego delle equazioni classiche dette *depth averaged shallow water*, trascurando gli effetti di stratificazione e focalizzando l'interesse su *barotropic features*. La soluzione numerica è stata ottenuta attraverso un metodo alle differenze finite [Finite Element Method - FEM] (Lalli et al, 2002)[1].

Grazie alle attuali elevate possibilità computazionali, l'interesse maggiore della comunità scientifica sulle tematiche dell'ingegneria costiera è oggi focalizzato su modelli 3-D (*idrostatici e non idrostatici*), in quanto, in molti casi, risulta importante predire le variazioni prodotte sul moto ondoso indotte da alterazioni di salinità e/o di temperatura. Da ciò, deriva la necessità di impiegare modelli 3-D[2]. Talvolta, però, la geometria del problema consente l'uso di equazioni mediate sulla profondità, che, quindi, sono ancora ampiamente usate con successo nelle applicazioni.

Il tema trattato si inquadra in un ambito più ampio che interessa anche numerosi altri effetti che concorrono a determinare uno dei fenomeni ambientali ad oggi di maggiore interesse, quello dello studio della dinamica dei litorali.

E' ad oggi, infatti, noto come le azioni del moto ondoso sotto costa e le correnti marine sono, in larga misura, la causa che determina l'evoluzione della linea di costa.
Fissando l'attenzione sulla fascia di territorio compresa tra la linea di battigia e la zona di frangimento, si ha che in corrispondenza della zona di frangimento l'onda è affetta da instabilità che ne produce la rottura, con conseguente alterazione del moto che da oscillatorio diventa traslatorio, cui è connessa una tendenza al trasporto di materiale solido. Ne deriva una complessa azione che interessa la trasformazione morfologica dei litorali (*comprendente sia le spiagge emerse che la piattaforma litoranea*), e che dipende, sostanzialmente, dai fenomeni di interazione tra il moto ondoso incidente, le caratteristiche morfologiche dei fondali e la granulometria del materiale solido sollecitato.

In questo complesso quadro si inseriscono anche i fenomeni di interazione onda-corrente, che determinano un'alterazione dei profili associati al moto ondoso e una modificazione degli effetti indotti su un litorale.

Il modellamento dei litorali, infatti, deriva principalmente dal trasporto litoraneo (*trasversale e longitudinale*) dei sedimenti ed è dovuto a due fenomeni distinti e contemporanei :
- la messa in sospensione dei sedimenti da parte del moto ondoso e/o dalle correnti;
- il generarsi di correnti marine, indotte dallo stesso moto ondoso, che traggono origine dal frangimento e costituiscono il veicolo di trasporto dei sedimenti.

Le correnti indotte sono di duplice natura :
- la *beach drifting*, localizzata nella zona compresa tra la linea di battigia e la linea di frangimento, ha un carattere oscillatorio, determinata da due movimenti contemporanei : un *flusso diretto*, generalmente obliquo, verso la riva, ed un *riflusso*, secondo le linee di massima pendenza del fondale, che produce uno spostamento delle particelle "*a dente*

[1] *In letteratura può essere reperito un numero molto grande di esempi di modelli numerici per la simulazi one di circolazione costiera : Foreman (1984), Casulli (1990), Borthwick and Barber (1992) Zhou e Stan sby (1999), Guillou e Nguyen (1999), Liska e Wendroff (1999)*
[2] *(cfr,, ad es., Balas e Ozhan (2000), Winters et al. (2000), Li e Wang (2000).*

di sega" ed uno spostamento del materiale solido parallelamente al litorale ed in direzione sottoflutto.

- la *longshore current*, che si produce in corrispondenza dei fondali in cui si verifica il frangimento e deriva dalla componente parallela alla riva dell'energia cinetica del moto ondoso, ed è anch'essa parallela alla costa.

Il complesso delle correnti indotte è soggetto a continua evoluzione, determinata dall'insorgere di interferenze con fattori esterni prodotti sia da cause naturali, che da fattori antropici. Per le prime, effetti rilevanti possono essere determinati da cambiamenti di direzione di corsi d'acqua, o da creazione di nuove foci, da cui si innescano azioni di contrasto con i flussi a superficie libera.

Tralasciando, per i secondi, gli aspetti più macroscopici e noti, come la costruzione di opere portuali o di difesa dei litorali, gli interventi di stabilizzazione dei versanti e la progressiva urbanizzazione delle zone costiere, appare non meno importante e significativa la scomparsa, a causa dell'inquinamento e della conseguente mancata ossigenazione delle acque, delle praterie marine di fanerogame, ed in genere della vegetazione nei bassi fondali litoranei, la cui assenza riduce la stabilizzazione del fondo marino ed esalta l'intensità del moto ondoso e, quindi, delle correnti indotte.

2. Calcolo numerico di flussi a superficie libera con il metodo degli elementi di contorno (Boundary Element Method – BEM): propagazione del moto ondoso

2.1 Il Modello Matematico

Nel presente paragrafo verrà analizzato, innanzitutto, il problema idrodinamico della propagazione del moto ondoso sia in condizioni di campo indisturbato che di propagazione in presenza di una struttura dalla geometria generica, nell'ipotesi che il moto sia incomprimibile ed irrotazionale.
Nel caso di fluido viscoso, il modello matematico capace di descrivere il moto nelle condizioni in esame è l'*equazione di Navier-Stokes*, che rappresenta la seconda legge della dinamica in termini della meccanica del continuo. Detta equazione stabilisce il seguente legame tra forze di volume, forze di superficie e accelerazione del fluido:

$$\rho\left(\frac{\partial\vec{u}}{\partial t}+\vec{u}\cdot\nabla\vec{u}\right)=-\nabla p+\rho\vec{g}+\mu\nabla^2\vec{u} \qquad (1)$$

in cui: ρ è la densità; $\vec{g}=(0,0,-g)$ è l'accelerazione di gravità; p è la pressione; μ è la viscosità; $\vec{u}=(u_x,u_y,u_z)$ è la velocità del fluido.

Per essere risolta l'equazione di Navier-Stokes deve essere accoppiata al *principio di conservazione della massa* espresso da

$$\nabla\cdot\vec{u}=0 \qquad (2)$$

e deve essere unita ad opportune *condizioni al contorno*.
L'applicazione del modello dell'equazione di Navier-Stokes presenta, però, alcuni problemi nell'ambito applicativo dell' ingegneria idraulica per i seguenti due motivi:

1. nelle applicazioni il numero di Reynolds $(\mathrm{Re}\equiv UL\rho/\mu)$ è generalmente elevato e può arrivare fino a valori di 10^9.
   I moti in cui il numero di Reynolds è così elevato sono caratterizzati dalla presenza di variazioni ad elevata frequenza, sia nello spazio che nel tempo, delle grandezze idrodinamiche, tali da richiedere griglie di calcolo molto fitte e, quindi, risorse di calcolo molto onerose. Conseguentemente la soluzione numerica del problema di Navier-Stokes in simulazione diretta (ossia senza introdurre modelli che tengano conto implicitamente di fenomeni turbolenti) risulta gravosa da un punto di vista computazionale.
   Una possibilità di risoluzione del problema è quella di risolvere il flusso con una griglia tale da descrivere le strutture idrodinamiche su grande scala, utilizzando dei modelli che tengano in qualche modo conto della distribuzione e della dissipazione di energia legata alle piccole strutture di flusso, nell'ipotesi che il comportamento del fluido su scale molto piccole, rispetto alla geometria del problema in esame, possa assumere carattere universale; il modo più semplice di operare in questa direzione consiste nel risolvere le equazioni del moto in termini di grandezze (opportunamen-

te) mediate, e modificare il coefficiente di viscosità per tenere conto delle dissipazioni aggiuntive;

2. la presenza della superficie libera, che è una grandezza variabile nel tempo, introduce delle grosse difficoltà in quanto richiede l'utilizzo di griglie di calcolo mobili che seguano l'evoluzione del moto ondoso al variare del tempo.

Risulta, pertanto, conveniente e preferibile l'utilizzo di modelli semplificati. A tal scopo pur mantenendo l'assunto di moto incomprimibile ed irrotazionale si farà riferimento nel proseguo ad un fluido non viscoso. I fenomeni inerenti la propagazione del moto ondoso risulteranno comunque analizzati nella loro globalità pur nella semplificazione che consente di trascurare gli effetti della viscosità.
Nell'ipotesi di assenza di viscosità, nell'equazione di Navier-Stokes (Eq. (1)) l'ultimo termine può essere trascurato, riducendosi alla cosiddetta equazione di Eulero.
Considerando, inoltre, il teorema di Kelvin secondo cui condizione necessaria e sufficiente affinché un moto piano sia irrotazionale è che (l'estensione al caso tridimensionale è immediata)

$$\frac{\partial u_y}{\partial z} = \frac{\partial u_z}{\partial y}$$

esiste una funzione detta "potenziale di velocità" $\Phi(x,y,z,t)$, tale che la velocità del fluido può essere espressa mediante la funzione potenziale di velocità

$$\vec{u}(x,y,z,t) = \nabla\Phi(x,y,z,t) \qquad (3)$$

L'introduzione di tale grandezza determina una notevole semplificazione del problema, infatti, il campo idrodinamico vettoriale risulta univocamente determinato da una funzione scalare.
Grazie alle semplificazioni introdotte, il problema del moto ondoso potrà essere totalmente studiato mediante il sistema di equazioni differenziali di Stokes (1847) valido per un moto irrotazionale a superficie libera.
Fissato un sistema di riferimento cartesiano *Oxyz*, con il piano *xy* orizzontale, coincidente con la superficie libera indisturbata, e l'asse *z* verticale orientato verso l'alto, le equazioni che costituiscono il sistema di Stokes è costituito dalle seguenti equazioni:

– *equazione di Bernoulli*, scritta nell'ipotesi di moto ideale, stazionario e irrotazionale, e particolarizzata alla quota $z = \eta$:

$$\frac{\partial\Phi}{\partial t} + \frac{1}{2}\nabla\Phi\cdot\nabla\Phi + g\eta = \frac{1}{\rho}f(t) \qquad \text{su } z=\eta \qquad (4)$$

Tale equazione esprime la condizione dinamica sulla superficie libera in quanto è stata ottenuta imponendo un valore costante della pressione sulla superficie libera, e, precisamente, imponendo che la pressione relativa alla pressione atmosferica sulla superficie libera dell'acqua sia nulla. Si osserva che tale assunto non consente di studiare, ad esempio, il problema della generazione di onde da parte del vento, in cui i gradienti di pressione in corrispondenza dell'interfaccia giocano un ruolo fondamentale.

Si osservi che questa relazione è stata ottenuta trascurando il contributo della tensione superficiale.

– *equazione cinematica sulla superficie libera*, che esprime, matematicamente, la condizione che la funzione $\eta(x,y,t)$ rappresenta l'elevazione istantanea della superficie libera dell'acqua, calcolata rispetto al livello di quiete:

$$\left(\frac{\partial\Phi}{\partial x}\right)_{z=\eta}\frac{\partial\eta}{\partial x}+\left(\frac{\partial\Phi}{\partial y}\right)_{z=\eta}\frac{\partial\eta}{\partial y}+\frac{\partial\eta}{\partial t}=\left(\frac{\partial\Phi}{\partial z}\right)_{z=\eta} \tag{5}$$

Tale condizione si ottiene individuando un opportuno volume di controllo, di ampiezza fissata, e di altezza individuata dall'elevazione della superficie libera in presenza di onda, per il quale si scrive l'equazione di continuità della massa liquida. L'acqua, che entra nel volume considerato, dovrà uguagliare l'acqua che esce attraverso il medesimo volume e quella che si accumula all'interno dello stesso.

– *equazione di continuità della massa*, che dà luogo all'equazione di Laplace:

$$\nabla^2\Phi=\frac{\partial^2\Phi}{\partial x^2}+\frac{\partial^2\Phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}=0 \tag{6}$$

che richiede, per la sua soluzione, l'imposizione di condizioni al contorno, che scaturiscono dal problema fisico in esame.

– *condizione al contorno* in corrispondenza di superfici rigide (fondale e/o strutture marine), supposte in stato di quiete, rappresentata dalla condizione di impermeabilità che analiticamente impone l'annullarsi della componente normale della velocità del fluido:

$$\frac{\partial\Phi}{\partial n}=0 \tag{7}$$

La normale $\vec{n}$ va intesa orientata verso l'interno del campo idrodinamico.

Il problema (4), (5), (6), (7) è non lineare per 2 motivi: le condizioni al contorno di frontiera libera contengono termini quadratici e, inoltre, devono essere soddisfatte su di una superficie che è, a sua volta, una incognita del problema. Oggi gli strumenti di calcolo disponibili consentono di risolvere problemi non lineari con buona accuratezza (Lalli et al., 1996); tuttavia, nel caso di onde di piccola ampiezza, può essere sufficiente risolvere il problema linearizzato, nel quale è possibile imporre le condizioni di frontiera libera sul piano z = 0 (superficie indisturbata). Nel seguito si farà riferimento proprio a tale formulazione.

2.1.1 Il Modello Matematico Linearizzato

Il problema della propagazione di onde di piccola ampiezza, sia in condizioni di campo indisturbato che di propagazione in presenza di una struttura dalla geometria generica, può essere rappresentato dalla forma linearizzata delle (4), (5), (6), (7):

$$\nabla^2\Phi = 0 \qquad \textit{nel campo idrodinamico D} \tag{8}$$

$$\frac{\partial\Phi}{\partial t} = -g\eta \qquad \text{su } z = 0 \tag{9}$$

$$\frac{\partial\eta}{\partial t} = \frac{\partial\Phi}{\partial z} \qquad \text{su } z = 0 \tag{10}$$

$$\frac{\partial\Phi}{\partial n} = 0 \qquad \textit{sul fondo} \tag{11}$$

Le condizioni linearizzate vengono imposte sulla superficie libera indisturbata (piano $z = 0$).

Nel caso di profondità costante *h* la soluzione più semplice per il problema cosi formulato è l'onda piana (che si propaga , ad esempio, lungo la direzione delle *x* crescenti) :

$$\eta(x,t) = A\cos(kx - \omega t) \tag{12}$$

caratterizzata da velocità di propagazione (o di fase)

$$c = \frac{\omega}{k} = \frac{\lambda}{T} \qquad \text{con} \begin{cases} k = \dfrac{2\pi}{\lambda} \\ \omega = \dfrac{2\pi}{T} \end{cases} \tag{13}$$

Il campo idrodinamico generato dalla propagazione di un'onda piana, avente elevazione d'onda di equazione (12), è, inoltre, caratterizzato dal potenziale di velocità :

$$\Phi(x,z,t) = \frac{gA}{\omega}\frac{\cosh[k(z+h)]}{\cosh(kh)}\sin(kx - \omega t) \tag{14}$$

Dal gradiente del potenziale [Eq. (3)] è possibile ottenere le componenti della velocità delle particelle fluide, mentre la velocità di propagazione dell'onda si calcola dalla relazione di dispersione:

$$c = \frac{\omega}{k} = \sqrt{\frac{g}{k}\tanh(kh)} \tag{15}$$

Quando si considera la propagazione di una pertubazione complessa, generata dalla sovrapposizione di onde piane come la (12), poiché il mezzo è dispersivo, è necessario tenere conto della velocità di gruppo:

$$C = \frac{d\omega}{dk} = c - \lambda \frac{dc}{d\lambda} \tag{16}$$

che è la velocità con cui si propaga l'energia associata ad un gruppo di onde. Ad esempio, quando si lancia un sasso in uno specchio d'acqua in quiete, si genera una pertubazione che si propaga in modo assialsimmetrico, coinvolgendo una zona sempre più estesa della superficie libera. Il fronte del sistema di onde (definito come il limite delle acque perturbate) si muove con velocità di gruppo; le armoniche componenti mantengono la propria individualità nell'ambito della pertubazione (si tenga presente che si tratta di un problema lineare) e si muovono con velocità di fase, maggiore della velocità di gruppo.
Se si pone attenzione al movimento di una cresta, si osserva che la sua forma si modifica nel tempo e, quando raggiunge il fronte, poiché la pertubazione complessiva si espande con velocità minore, scompare. E' semplice verificare che, nel caso di acque profonde, poiché $\tanh(kh) \to 1,$ la velocità di gruppo è proprio metà della celerità dell'onda, ovvero $C = c/2$.

Per studiare, mediante simulazione numerica, l'evoluzione di un sistema ondoso come quello finora descritto, è opportuno esprimere il problema in termini adimensionali. Siano:

- h la profondità;
- T il periodo dell'onda;
- A l'ampiezza dell'onda;
- λ la lunghezza dell'onda.

Si introducono le seguenti variabili adimensionali (le variabili dimensionali vengono indicate con apice) :

$$x = \frac{x'}{h} \qquad z = \frac{z'}{h} \qquad t = \frac{t'}{T} \qquad \Phi = \Phi' \frac{T}{h^2} \qquad \lambda = \frac{\lambda'}{h} \tag{17}$$

nonchè i due parametri adimensionali:

$$\varepsilon = \frac{A}{h} \qquad \Omega = \frac{gT^2}{h} \tag{18}$$

tramite cui le equazioni lineari del sistema di Stokes risolutive del problema possono essere riscritte in forma dimensionale; Eqs. (9), (10), (9) e (11) rispettivamente come segue

$$\frac{\partial \Phi}{\partial t} = -\Omega \eta \qquad x > 0,\ z = 0 \tag{19}$$

$$\frac{\partial \eta}{\partial t} = \frac{\partial \Phi}{\partial z} \qquad x > 0,\ z = 0 \tag{20}$$

$$\nabla^2 \Phi = 0 \qquad x > 0 \tag{21}$$

$$\frac{\partial \Phi}{\partial z} = 0 \qquad x > 0,\ z = -1 \tag{22}$$

La lunghezza dell'onda generata sarà uguale a :

$$\lambda = \Omega \frac{\tanh(2\pi/\lambda)}{2\pi} \tag{23}$$

con i due casi limite:

$$\begin{cases} \lambda_\infty = \dfrac{\Omega}{2\pi} & acque\ profonde \quad (\Omega \to 0) \\ \lambda_0 = \sqrt{\Omega} & acque\ basse \quad (\Omega \to \infty) \end{cases} \tag{24}$$

2.2 Il Modello Discreto.

2.2.1 Cenni di teoria del potenziale

La teoria del potenziale consente di esprimere un problema differenziale mediante una formulazione integrale, il che comporta dei vantaggi:

- la formulazione integrale diminuisce di uno la dimensione del dominio del problema in esame: in problemi 3-D vengono prese in considerazione delle superfici anziché dei volumi, mentre in problemi 2-D delle linee anziché delle superfici, con ovvi vantaggi dal punto di vista della soluzione numerica;

- la teoria delle equazioni integrali fornisce strumenti matematici atti a delineare strategie per la formulazione di teoremi di esistenza e unicità della soluzione.

In un problema 3-D, definito in un dominio $D + \partial D$ (essendo ∂D la frontiera) si può introdurre il *potenziale superficiale di semplice strato*:

$$\Phi(x,y,z) = \int_{\partial D} \frac{\sigma(\tilde{x},\tilde{y},\tilde{z})}{\sqrt{(x-\tilde{x})^2 + (y-\tilde{y})^2 + (z-\tilde{z})^2}} dS(\tilde{x},\tilde{y},\tilde{z}) \tag{25}$$

mentre, in un problema 2-D si ha:

$$\Phi(x,z) = \int_{\partial D} \sigma(\tilde{x},\tilde{z}) \log\sqrt{(x-\tilde{x})^2 + (z-\tilde{z})^2}\, dS(\tilde{x},\tilde{z}) \tag{26}$$

nelle quali σ è la *densità di distribuzione del potenziale sulla superficie* (o sulla superficie sulla linea) ∂D. Il potenziale di strato semplice è continuo nei punti di ∂D: si può infatti dimostrare che negli integrali (25, 26) convergono uniformemente.

La derivata normale del potenziale superficiale di semplice strato contiene una discontinuità, della quale è necessario tenere opportunamente conto nell'imposizione della condizione

al contorno di impermeabilità (11). In base alle condizioni al contorno che la funzione deve soddisfare, si hanno i cosiddetti problemi *ai limiti*:

- problema interno (o esterno) di Dirichelet : consiste nel determinare una funzione Φ
  1. definita continua nel dominio limitato interno (o illimitato esterno) $D + \partial D$
  2. verifica nel campo D l'equazione di Laplace $\nabla^2 \Phi = 0$.
  3. assume su ∂D dei valori assegnati.

- problema interno (o esterno) di Neumann: consiste nel determinare una funzione Φ
  1. definita continua nel dominio limitato interno (o illimitato esterno) $D + \partial D$
  2. verifica nel campo D l'equazione di Laplace $\nabla^2 \Phi = 0$.
  3. la sua derivata normale $\partial \Phi / \partial n$ assume su ∂D dei valori assegnati.

Ad esempio, la soluzione di un problema di Neumann, formulato mediante il potenziale di strato semplice, implica la soluzione numerica di un'equazione integrale di Fredholm di II specie, per la quale sono stati dimostrati teoremi di esistenza ed unicità. Nel caso 2-D si ottiene:

$$\frac{\partial \Phi(x,z)}{\partial n} = \pi\sigma(x,z) + \int_{\partial D} \sigma(\tilde{x},\tilde{z}) \frac{\partial}{\partial n} \log \sqrt{(x-\tilde{x})^2 + (z-\tilde{z})^2}\, dS(\tilde{x},\tilde{z}) \qquad x,z \in \partial D \quad \textbf{(27)}$$

Poiché il potenziale, così definito ed espresso dalla Equazioni (25) e (26), soddisfa automaticamente l'equazione di Laplace, *il problema si riduce alla determinazione della funzione σ in modo che le condizioni al contorno siano verificate*. Ecco, perché, utilizzando una formulazione integrale al contorno si può risolvere un problema operando unicamente sulla frontiera del dominio.

Inoltre, si ha che:

- nella *formulazione integrale classica* la frontiera del dominio fisico coincide con la superficie (o con la linea nel caso 2-D) sede della distribuzione del potenziale di semplice strato;

- nella *formulazione cosiddetta desingolarizzata*, per contro, il potenziale viene distribuito su di una superficie (o linea) esterna al campo idrodinamico, posta ad una distanza opportuna dalla frontiera del problema fisico.

Quest'ultimo tipo di approccio, della formulazione cosiddetta desingolarizzata, comporta delle notevoli semplificazioni:

1. si elimina il problema della discontinuità della derivata normale del potenziale
2. si evita il calcolo di integrali singolari

3. il potenziale di semplice strato può essere approssimato, nell'ambito della procedura numerica di calcolo, con una distribuzione discreta di *N* sorgenti puntiformi localizzate esternamente al campo idrodinamico.

Si ottiene pertanto (problema 3-D):

$$\Phi(x,y,z)=\sum_{k=1}^{N}\frac{\sigma_k}{\sqrt{(x-\tilde{x}_k)^2+(y-\tilde{y}_k)^2+(z-\tilde{z}_k)^2}} \tag{28}$$

oppure (problema 2-D):

$$\Phi(x,z)=\sum_{k=1}^{N}\sigma_k \log\sqrt{(x-\tilde{x}_k)^2+(z-\tilde{z}_k)^2} \tag{29}$$

nelle quali *N è il numero di sorgenti utilizzato*. Il potenziale espresso dalla (28) oppure dalla (29) soddisfa ancora l'equazione di Laplace e, con semplici operazioni, si possono ottenere le sue derivate fino all'ordine desiderato.

La soluzione numerica di un problema di flusso potenziale si riduce quindi alla ricerca dei valori σ_k, che si ottengono imponendo le condizioni al contorno.

Per risolvere numericamente il problema della propagazione del moto ondoso si esprime il potenziale Φ per mezzo della (29).
A tale scopo si definiscono le *matrici di influenza* che esprimono il potenziale e le due componenti di velocità indotti nel punto (x_i, z_i) da una sorgente puntiforme di intensità unitaria collocata nel punto $(\tilde{x}_k, \tilde{z}_k)$:

$$\varphi(i,k)=\log\sqrt{\sqrt{(x_i-\tilde{x}_k)^2+(z_i-\tilde{z}_k)^2}}=\frac{1}{2}\log\left[(x_i-\tilde{x}_k)^2+(z_i-\tilde{z}_k)^2\right] \tag{30}$$

$$\varphi_x(i,k)=\frac{x_i-\tilde{x}_k}{(x_i-\tilde{x}_k)^2+(z_i-\tilde{z}_k)^2} \tag{31}$$

$$\varphi_z(i,k)=\frac{z_i-\tilde{z}_k}{(x_i-\tilde{x}_k)^2+(z_i-\tilde{z}_k)^2} \tag{32}$$

La determinazione della ubicazione ottimale di tali punti costituisce un problema piuttosto delicato, legato all'accuratezza della soluzione; in libera di massima, i punti $\tilde{x}_k, \tilde{z}_k$ devono avvicinarsi al contorno fisico del problema man mano che la griglia di calcolo viene infittita. Si può scegliere una distanza dal contorno pari a $\sqrt{\Delta s}$, con $\Delta s=\left|\tilde{P}_k-\tilde{P}_{k-1}\right|=\left|P_i-P_{i-1}\right|$ (per ulteriori dettagli vedi Lalli et al., 1996).

2.2.2 Soluzione numerica nel caso lineare di moto ondoso che si propaga in presenza di una secca

Per risolvere il sistema, nel caso di moto ondoso che si propaga in presenza di una struttura, viene adottato il metodo della perturbazione, il quale consiste nello scomporre le funzioni incognite, elevazione d'onda $\eta(x,y,t)$ e potenziale di velocità $\Phi(x,y,z,t)$, nella somma di due termini relativi rispettivamente alle condizioni dell'onda incidente e alla condizione di perturbazione indotta dalla presenza dell'ostacolo.
Il potenziale di velocità potrà, perciò, essere scritto come:

$$\Phi = \phi + \tilde{\phi} \tag{33}$$

in cui

- ϕ è il potenziale associato all'onda incidente, che è noto;
- $\tilde{\phi}$ è il potenziale di perturbazione indotto dall'ostacolo.

Analogamente si avrà un termine di elevazione d'onda associato all'onda incidente, η, ed un altro relativo alla perturbazione indotta dall'ostacolo, $\tilde{\eta}$.

Avendo introdotto questa scomposizione delle grandezze del campo di moto, la soluzione del problema è determinata, considerando che:

1. il potenziale associato all'onda incidente, ϕ, che è noto, deve essere soluzione del seguente problema:

$$\nabla^2 \phi = 0 \qquad \textit{nel campo idrodinamico D} \tag{34}$$

$$\frac{\partial \phi}{\partial t} = -g\eta \qquad \text{su } z = 0 \tag{35}$$

$$\frac{\partial \eta}{\partial t} = \frac{\partial \phi}{\partial z} \qquad \text{su } z = 0 \tag{36}$$

$$\frac{\partial \phi}{\partial n} = 0 \qquad \textit{sui contorni solidi} \tag{37}$$

2. il potenziale di perturbazione indotto dall'ostacolo, $\tilde{\phi}$, deve soddisfare le seguenti equazioni:

$$\nabla^2 \tilde{\phi} = 0 \qquad \textit{nel campo idrodinamico D} \tag{38}$$

$$\frac{\partial \tilde{\phi}}{\partial t} = -g\tilde{\eta} \qquad \text{su } z = 0 \tag{39}$$

$$\frac{\partial \tilde{\eta}}{\partial t} = \frac{\partial \tilde{\phi}}{\partial z} \qquad \text{su } z = 0 \tag{40}$$

$$\frac{\partial \tilde{\phi}}{\partial n} = -\frac{\partial \phi}{\partial n} \qquad \textit{sui contorni solidi} \tag{41}$$

Si osserva, infine, che devono essere assegnate le condizioni iniziali per la grandezze associate alla perturbazione, ovvero $\tilde{\phi}$ ed $\tilde{\eta}$ per le quali si fissa in tutte le computazioni che all'istante iniziale $t = 0$:

$$\tilde{\phi}(x, y, z, 0) = 0 \qquad \tilde{\eta}(x, y, 0) = 0 \tag{42}$$

Come è stato, precedentemente detto, infatti il potenziale di velocità e l'elevazione d'onda associati all'onda incidente, ϕ ed η, sono noti mentre le rispettive grandezze $\tilde{\phi}$ ed $\tilde{\eta}$ relative alla condizione di perturbazione indotta dall'ostacolo sono incognite. Per quanto riguarda il potenziale di perturbazione, $\tilde{\phi}$, questo può essere espresso mediante la teoria del potenziale come:

$$\tilde{\phi}(P) = \int_{\partial D} \sigma(P^*) G(P, P^*) dS^* \tag{43}$$

in cui:

- punto $P = P(x, y, z)$ in si calcola il potenziale indotto da una sorgente puntiforme di intensità unitaria collocata nel punto $P^* = P^*(x^*, y^*, z^*)$;
- $G(P, P^*) = \dfrac{1}{|P - P^*|}$;
- σ è la *densità di distribuzione del potenziale sulla superficie* ∂D.
- ∂D rappresenta la superficie integrale che è estesa all'intero dominio fisico considerato.

Si osserva che nel caso lineare esaminato, la discretizzazione della superficie del corpo solido non è complessa in quanto la superficie libera si suppone essere coincidente con il livello di medio mare. Perciò, la superficie bagnata del corpo solido non cambia al variare del tempo. Al fine di effettuare la soluzione numerica del problema, inoltre, la superficie dell'ostacolo ∂O (in questo caso la secca posta sul fondo), è stata discretizzata in *N* pannelli piani a forma di quadrilatero, sui quali la densità superficiale di distribuzione del potenziale si può considerare quasi costante (Hess and Smith, 1966).
Inoltre, l'integrale (43) è stato applicato nella sua formulazione desingolarizzata, come suggerito da Cao et al. (1991), secondo cui, sulla superficie libera *S*, si suppone essere uniformemente distribuite *M* sorgenti puntiformi localizzate ad una distanza fissata dalla superficie libera.
Infine, la presenza del fondo è tenuta in considerazione attraverso il metodo delle immagini.

In definitiva, quindi, il potenziale di perturbazione, precedentemente menzionato e rappresentato dall'Equazione (43), può essere espresso mediante nel seguente modo:

$$\tilde{\phi}(P) = \sum_{j=1}^{N} \sigma_j \left[\int_{\partial O_j} G(P, P^*) dS^* + \int_{\partial \overline{O}_j} G(P, P^*) dS^* \right] + \sum_{k=1}^{N} \overline{\sigma}_k \left[G(P, P_k) + G(P, \overline{P}_k) \right] \quad \textbf{(44)}$$

in cui

- ∂O_j è il *j*-esimo pannello con cui è stata discretizzata la superficie della secca posta sul fondo;
- σ_j è la densità di distribuzione del potenziale sulla superficie del *j*-esimo pannello con cui è stata discretizzata la superficie della secca posta sul fondo; si suppone che tale densità sia costante per tutti i pannelli della secca;
- $\partial \overline{O}_j$ è l'immagine di ∂O_j rispetto al fondo;
- P_k è la posizione della *k*-esima sorgente posta al di sopra della superficie libera;
- $\overline{P}_k$ è l'immagine di P_k.

Va tenuto, infine, presente che il modello descritto è di natura non stazionaria, in quanto sono presenti derivate temporali; è necessario pertanto utilizzare uno schema numerico atto a seguire l'evoluzione del fenomeno nel tempo.
In altri termini, stabilito un passo temporale Δt, si vuole conoscere il valore delle grandezze idrodinamiche in tutti gli istanti successivi.
E precisamente si procederà nel seguente modo: noti i valori della soluzione all'istante t^n si determineranno i valori della soluzione all'istante successivo $t^{n+1} = t^n + \Delta t$ mediante lo schema classico di Runge-Kutta al quarto ordine.
Per ogni passo intermedio di Runge-Kutta lo schema è calcolato come segue:

- dato il potenziale di perturbazione $\tilde{\phi}$ e le sue derivate sia sulla superficie della secca posta sul fondo, ∂O, sia sulla superficie libera, S, all'istante precedente. Nota questa grandezza è possibile aggiornare la condizione sia dinamica che cinematica sulla superficie libera, ovvero le equazioni rispettivamente (40) e (39), e calcolare, perciò, il valore del potenziale di velocità e della superficie libera su S;

- *i nuovi valori di* $\tilde{\phi}$ [associati alle *M* sorgenti puntiformi distribuite alla distanza fissata dalla superficie libera] e *le condizioni al contorno del problema di Neumann* (precedentemente discusso) [espresse dall'equazione (44) calcolata sulle *N* sorgenti poste sulla superficie della secca] determinano *N* + *M* condizioni lineari relative alle *N* + *M* incognite σ_j e $\tilde{\sigma}_k$. Il sistema lineare risultante è risolto mediante fattorizzazione LU della matrice dei coefficienti;

- le nuove densità di distribuzione superficiali del potenziale, σ_j e $\tilde{\sigma}_k$, ottenute al passo precedente, consentono di determinare i valori di $\tilde{\phi}$ e delle sue derivate sia sulla superficie della secca posta sul fondo, ∂O, sia sulla superficie libera, S, al passo successivo di Runge-Kutta.

2.2.2.1 Smorzamento dell'onda

Allo scopo di evitare la riflessione dell'onda sul contorno non fisico della superficie libera, viene applicato il modello della linea di smorzamento suggerita da Baker et al. (1989).
Tale modello consiste nell'aggiungere il termine $-\nu\tilde{\phi}$ a secondo membro della condizione dinamica al contorno sulla superficie libera, rappresentata dall'equazione (39); mentre il termine $-\nu\tilde{\eta}$ è introdotto a secondo membro della condizione cinematica al contorno sulla superficie libera, rappresentata dall'equazione (40).
In definitiva, allo scopo di produrre uno smorzamento dell'eventuale onda riflessa dall'interazione con l'ostacolo, le condizioni cinematica e dinamica [Eq. (40) e (39)] sulla superficie libera assumono rispettivamente le seguenti forme:

$$\frac{\partial\tilde{\eta}}{\partial t} = \frac{\partial\tilde{\phi}}{\partial z} - \nu\tilde{\eta} \qquad \text{su } z = 0 \tag{45}$$

$$\frac{\partial\tilde{\phi}}{\partial t} = -g\tilde{\eta} - \nu\tilde{\phi} \qquad \text{su } z = 0 \tag{46}$$

in cui:

- ν è un coefficiente che dipende dalla distanza dall'ostacolo.

Il coefficiente ν deve essere opportunamente calibrato con esperimenti numerici, al fine di minimizzare la riflessione all'estremità del dominio di calcolo senza che gli effetti dissipativi disturbino, in una zona sufficientemente ampia, il flusso generato dalla parte oscillante.
In particolare, nel caso in esame, tale coefficiente ν assume la seguente espressione:

$$\nu = \begin{cases} 0 & R < R_0 \\ \nu_0 \left(\dfrac{R - R_0}{R_1 - R_0} \right)^2 & R_0 \le R \le R_1 \\ \nu_0 & R > R_1 \end{cases} \tag{47}$$

in cui:

- ν_0 è un coefficiente fissato (tipicamente $\nu_0 = O(1)$ nelle applicazioni);
- $R = \sqrt{x^2 + y^2}$ è la distanza dall'origine;
- R_0 e $R_1 > R_0$ sono due numeri reali scelti sulla base dell'esperienza numerica $(R_1 - R_0 = O(\lambda))$.

Nel campo lontano dalla secca i termini aggiuntivi (che figurano nelle equazioni (45) e (46)) fungono da smorzamento solo sul potenziale di perturbazione $\tilde{\phi}$ e sull'elevazione d'onda $\tilde{\eta}$, mentre non sono affetti da smorzamento le condizioni di moto iniziali, assegnate mediante le funzioni ϕ e η. E questo è proprio uno più grandi vantaggi della decomposizione (33) effettuata.

2.2.2.2 Geometria della secca posta sul fondo

Oggetto delle applicazioni effettuate è stato l'analisi degli effetti indotti dalla diffrazione/riflessione prodotti da una secca dalla forma semplice.
Per quanto riguarda la geometria della secca questa coincide con quella delineata in un esperimento idraulico condotto da Ito e Tanimoto (1972); in particolare le equazioni che individuano tale secca sono:

$$z = \begin{cases} (h_M - h_m)\left(\dfrac{x^2 - y^2}{R^2}\right) + h_m & x^2 + y^2 < R^2 \\ h_M & x^2 + y^2 \geq R^2 \end{cases} \qquad (48)$$

in cui:

- h_M e h_m sono rispettivamente la profondità massima e quella minima;
- R l'estensione radiale della secca.

3. Calcolo numerico con il metodo degli elementi degli elementi finiti (Finite Element Method – FEM): interazione di onde con un flusso viscoso e non stazionario

3.1 Il Modello Matematico di shallow-water

Come inizialmente detto, in molte situazioni, i flussi a superficie libera sono caratterizzati da fenomeni di interazione onde-correnti. Getti e corsi d'acqua possono essere fortemente interessati dall'interazione con onde di mare, e contemporaneamente il profilo del moto ondoso può essere significatamene modificato dalle correnti.

In un recente lavoro di Teng et al. (2001) è stato proposto un modello per flusso potenziale interamente 3-D per l'analisi e la previsione della diffrazione del moto ondoso indotta da una secca (shoal) e per l'interazione di onde di mare con correnti. Tale approccio è lineare dal momento che la funzione di Green, inerente la formulazione integrale, soddisfa le condizioni al contorno lineari della superficie libera; inoltre, in questo lavoro l'interazione di onde mare con correnti è limitata al caso di flusso stazionario.
In molte situazioni di pratico interesse nell'ambito ambientale, tuttavia, vi sono diverse configurazioni che implicano un flusso viscoso non uniforme, come nel caso di interazione onda di mare con un getto.
Un approccio 3-D sia totalmente non lineare sia lineare, non stazionario, per un flusso potenziale a superficie libera è stato proposto da Lalli et al. (1996) e da Lalli (1997), ed è stato oggetto di discussione nei precedenti paragrafi. Nei lavori citati la velocità del flusso potenziale è stato separata nelle componenti di flusso potenziale incidente e di flusso potenziale perturbato (Landrini, 1994) [si tratta del metodo esposto nel paragrafo 2.2.2] e rappresentato attraverso una distribuzione di strato semplice sulle condizioni al contorno del dominio del fluido (Bassanini et al., 1994).

Partendo da tali risultati, si vuole adesso proporre un metodo numerico basato sul metodo numerico delle differenze finite (Finite Element Method - FEM) per lo studio dei problemi idrodinamici dell'interazione onde di mare con correnti nel dominio del tempo. Si considera, inoltre, anche una configurazione in cui sia presente una struttura dalla geometria assegnata; nel caso specifico si considererà la presenza di una secca sul fondo (la geometria semplificata di tale struttura è analoga a quella delineata nel paragrafo 2.2.2.2). Perciò, il modello descritto si propone di analizzare l'evoluzione del moto ondoso che si propaga in presenza di un flusso (una corrente), che può essere viscoso e non stazionario, e che interagisce con una secca posta sul fondo [il modello può essere applicato anche nel caso in cui sia presente una struttura dalla generica geometria].

Si assuma la densità del fluido uniforme, e sia $\underline{u}(x,y,z,t)$ un campo di velocità rotazionale e non stazionaria, che in accordo con la decomposizione di Helmholtz, può essere scritto come:

$$\underline{u}(x,y,z,t) = \overline{\underline{u}}(x,y,z,t) + \nabla\Phi(x,y,z,t) \tag{49}$$

$$\Gamma(x,y,t) = \overline{\eta}(x,y,t) + \eta(x,y,t) + \tilde{\eta}(x,y,t) \tag{50}$$

in cui

- Γ è la funzione elevazione d'onda;
- $\underline{\overline{u}}(\overline{u},\overline{v})$ è la velocità mediata sulla profondità;
- $\overline{\eta}$ è la superficie libera mediata sulla profondità;
- $\nabla\Phi = \nabla\varphi + \nabla\tilde{\varphi}$ è il flusso potenziale totalmente 3-D, scritto mediante il metodo di decomposizione delineato nel paragrafo 2.2.2;
- φ è il flusso potenziale associato all'onda incidente;
- η è la superficie libera associata all'onda incidente;
- $\tilde{\varphi}$ è il flusso potenziale di perturbazione prodotto dall'interazione onda-struttura-corrente;
- $\tilde{\eta}$ è la superficie libera di perturbazione prodotto dall'interazione onda-struttura-corrente.

Come evidenziato dalle equazioni (49) e (50), perciò, nel seguente approccio, gli effetti viscosi sono descritti attraverso un modello mediato sulla profondità (depth-averaged model), mentre gli effetti della superficie libera sono considerati attraverso un modello a flusso potenziale totalmente 3-D.

Si consideri, infatti, che se una configurazione di flusso potenziale (vedi ad esempio. Clercx et al., 2003) può essere studiata mediante un approccio 2-D, per quanto riguarda, invece, la componente viscosa, la propagazione del moto ondoso è un fenomeno tipicamente 3-D.

Per il caso di onde elementari di piccola ampiezza, il potenziale dell'onda incidente è dato dalla soluzione del problema di Stokes, che nel caso lineare è definito dalle equazioni (8), (9), (10) e (11), e che in funzione delle grandezze in esame può essere espresso come segue:

$$\frac{\partial\varphi}{\partial t} + g\eta = 0 \qquad \text{su } z = 0 \tag{51}$$

$$\frac{\partial\eta}{\partial t} = \frac{\partial\varphi}{\partial z} \qquad \text{su } z = 0 \tag{52}$$

$$\frac{\partial\varphi}{\partial z} = 0 \qquad \text{su } z = -h_0 \tag{53}$$

L'onda incidente è poi espressa come una combinazione di onde lineari. Gli effetti prodotti da variazioni di profondità nel dominio del fluido e da ogni tipo di ostacolo possono essere tenuti in considerazione mediante il potenziale di perturbazione $\tilde{\varphi}$. L'estensione alla formulazione non lineare è diretta (Lalli et al., 1996; Lalli, 1997), sebbene non sia banale da un punto di vista della computazione. In questo caso, i problemi più ardui riguardano il frangimento delle onde. E', infatti, importante evidenziare come il presente approccio non permette, come principio, la simulazione del frangimento delle onde (vedi per esempio Guignard et al., 1999), poiché il frangimento fornisce un flusso turbolento totalmente 3-D e

questo argomento è fuori dallo scopo della presente trattazione.

Per quanto riguarda il campo di moto rotazionale considerato: le velocità, $\underline{\overline{u}}$, e l'elevazione d'onda, $\overline{\eta}$, mediate sulla profondità è lecito assumere che soddisfino le seguenti *equazioni di shallow-water nella forma conservativa*:

$$\frac{\partial \underline{\overline{u}}}{\partial t} + \nabla\left[\frac{1}{2}\left|\underline{\overline{u}}\right|^2 + g\overline{\eta}\right] = \underline{\overline{u}} \times \omega + \nu\nabla^2\underline{\overline{u}} \tag{54}$$

$$\frac{\partial \overline{\eta}}{\partial t} + \frac{\partial\left[\left(h+\overline{\eta}\right)\overline{u}\right]}{\partial x} + \frac{\partial\left[\left(h+\overline{\eta}\right)\overline{v}\right]}{\partial y} = 0 \tag{55}$$

nelle quali:

- $\omega = \nabla \times \underline{\overline{u}} = \nabla \times \underline{u}$ è la vorticità;
- $h = h(x, y)$ è la profondità.

Allo scopo di ottenere un'idonea formulazione per l'interazione onde di mare con correnti, tenendo in considerazione le Equazioni (49), (50), (54) e (55) e la conservazione del momento sulla superficie libera, si ottiene che:

$$\nabla\left[\frac{\partial\Phi}{\partial t} + \frac{1}{2}\left|\nabla\Phi\right|^2 + \underline{\overline{u}}\,\nabla\Phi + g\left(\eta + \tilde{\eta}\right)\right] = \nabla\Phi \times \omega \qquad \text{su } z = \Gamma \tag{56}$$

Trascurando i termini non lineari e considerando l'Equazione (51) si ottiene che:

$$\nabla\left[\frac{\partial\tilde{\varphi}}{\partial t} + \underline{\overline{u}}\,\nabla\left(\varphi + \tilde{\varphi}\right) + g\tilde{\eta}\right] = \nabla\Phi \times \omega \qquad \text{su } z = 0 \tag{57}$$

La condizione cinematica sulla superficie libera, nel caso esaminato, è espressa come:

$$\frac{\partial\Gamma}{\partial t} + \underline{u}\frac{\partial\Gamma}{\partial x} + \underline{v}\frac{\partial\Gamma}{\partial y} = \underline{w} \qquad \text{su } z = \Gamma \tag{58}$$

Effettuando la linearizzazione e considerando le Equazioni (52) (54) e (55) si ottiene:

$$\frac{\partial\tilde{\eta}}{\partial t} + \underline{u}\left(\frac{\partial\eta}{\partial x} + \frac{\partial\tilde{\eta}}{\partial x}\right) + \underline{v}\left(\frac{\partial\eta}{\partial y} + \frac{\partial\tilde{\eta}}{\partial y}\right) = \tilde{\varphi}_z + \left(\frac{\partial h\overline{u}}{\partial x} + \frac{\partial h\overline{v}}{\partial y}\right) + \overline{\eta}\left(\frac{\partial\overline{u}}{\partial x} + \frac{\partial\overline{v}}{\partial y}\right) \qquad \text{su } z = 0 \tag{59}$$

Ottenuta la soluzione del problema $\nabla^2\tilde{\varphi} = 0$ attraverso le condizioni al contorno [(Equazioni (57) e (59)] e l'equazione sui contorni solidi: $\tilde{\varphi}_n = -\varphi_n$, è, quindi, possibile determinare la componente del flusso viscoso attraverso le Equazioni (54) e (55) con le opportune condizioni al contorno, consentendo di determinare la simulazione del flusso onda di mare corrente nel dominio del tempo. L'ipotesi di base dell'approccio proposto è che tutti gli effetti di in-

terazione possono essere descritti attraverso il potenziale di perturbazione $\tilde{\varphi}$; tale assunto assomiglia all'ipotesi di Taylor relativa, come è noto, al concetto della cosiddetta turbolenza di congelamento (vedi ad esempio McComb, 2000).

Il campo viscoso, non stazionario, mediato sulla profondità è ottenuto utilizzando un opportuno modello a viscosità turbolenta (Nezu e Nakagawa, 1993; Spalart e Allmaras, 1994), e la simulazione numerica è ottenuta attraverso un metodo alla differenze finite descritto nel precedente lavoro (Lalli et al., 2002; Le e Moin, 1991).

Dall'altro canto, il potenziale del campo di moto totalmente 3-D è computato nel dominio del tempo mediante una formulazione integrale desingolarizzata al contorno, il metodo numerico è delineato nei lavori Lalli et al. (1996) e Lalli (1997), nei quali l'efficienza e l'accuratezza dei metodi integrali al contorno nella forma desingolarizata sono mostrati e specificati.

3.2 Analisi dei risultati determinati mediante il modello matematico proposto: caso di onde lineari stazionarie generate da un ostacolo immerso in una corrente uniforme

Come commento alla decomposizione del flusso e al modello presentati, potrebbe essere vantaggioso mostrare (Figura *1a* e *1b*) le soluzioni del flusso potenziale 2-D con quello della superficie libera generato dall'interazione di una corrente uniforme con una secca.
Tale configurazione può essere analizzata mediante le equazioni dello shallow-water (acque basse) per quanto riguarda il flusso uniforme e mediante un modello di moto potenziale totalmente 2-D per quanto riguarda il moto ondoso.

In particolare, le equazioni 1-D dello shallow-water, linearizzate rispetto al flusso libero uniforme U_∞ e assumendo $\left(h+\overline{\eta}\right)\overline{u} \approx h\overline{u}$ (lecita nel caso analizzato di onde di piccola ampiezza), determinano la forma della superficie libera:

$$\frac{\overline{\eta}}{h_0} = -Fr^2 \frac{1-h(x)/h_0}{h(x)/h_0 - Fr^2} \tag{60}$$

con, ovviamente, $Fr = U_\infty / \sqrt{gh_0}$ il numero di Froude.
Il flusso potenziale 2-D, d'altro canto, è calcolato attraverso le condizioni lineari della superficie libera, il cosiddetto problema di Neumann-Kelvin.

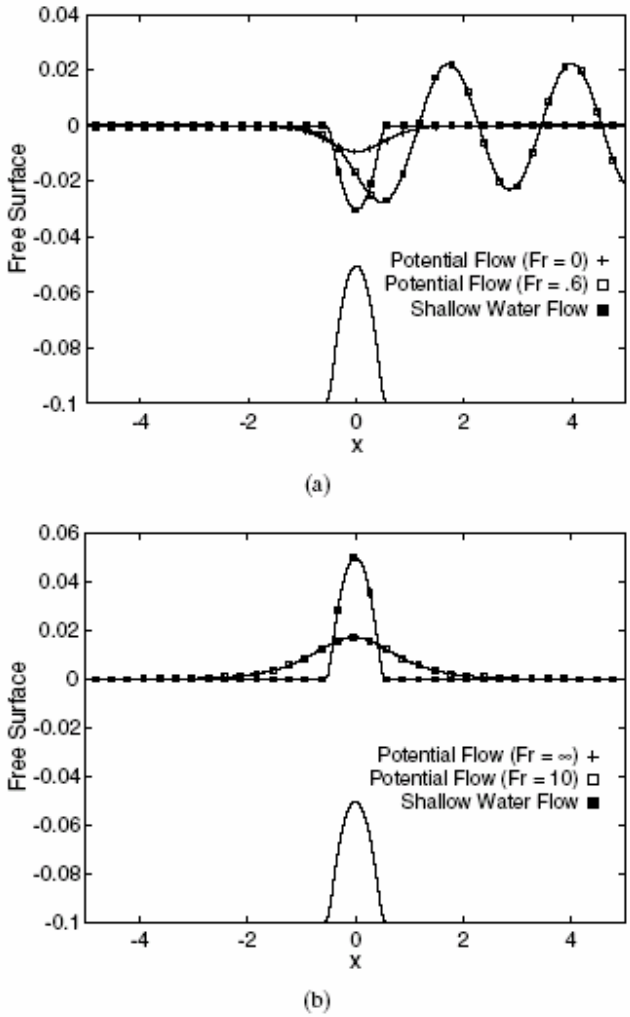

Fig. 1 (a) Interazione tra una corrente uniforme e una secca: elevazione d'onda (Fr=0.6).
(b) Interazione tra una corrente uniforme e una secca: elevazione d'onda (Fr=10).

La Figura *1a* mostra, per il caso sub-critico, come il modello per shallow-water porta ad ottenere una forma della superficie libera simile, quantitativamente, a quella ottenuta mediante un flusso 2-D con fondo rigido, come ad esempio il flusso con numero di Froude nullo, che può essere facilmente computato mediante il metodo delle immagini. Infatti, l'approssimazione idrostatico non permette la generazione del treno di onde, che, invece, può essere ottenuta mediante la soluzione del flusso potenziale totalmente 2-D che soddisfa le condizioni lineari al contorno per la superficie libera.
Per il caso super-critico, le cose sono molto differenti, per il quale il treno di onde non può essere generato (Figura *1b*), e sia l'equazione dello shallow-water che quella del flusso potenziale 2-D danno gli stessi risultati. In questo caso, la forma della superficie libera per $Fr = \infty$

è stata ottenuta mediante il metodo delle immagini negative ed è all'incirca coincidente con la soluzione per $Fr = 10$.

Per quanto riguarda la linearizzazione del modello idrostatico di shallow-water, l'equazione (60) chiaramente indica che, per grandi numeri di Froude, la forma della superficie libera è parallela al fondo (Figura *1b*).
Questo semplice esempio mostra in che senso i modelli mediati sulla profondità, basati sull'approssimazione idrostatica, non sono idonei per la simulazione di flussi a superficie libera, tranne il caso in cui il flusso sia super-critico.
Queste configurazioni sono state dimostrate e ottenute mediante modelli di Boussinesq e formulazioni mediate sulla profondità (per esempio vedi Mei, 1982).

L'accuratezza dello schema numerico proposto è testata in un caso molto semplice, per il quale la soluzione analitica è disponibile, quale è ad esempio l'interazione tra un'onda periodica lineare $\eta = A_0 \cos(k_0 x - \sigma t)$ e una corrente uniforme; in questo caso le Equazioni (58) e (59) diventano:

$$\frac{\partial \tilde{\varphi}}{\partial t} + U_\infty \left(\varphi_x + \tilde{\varphi}_x\right) + g\tilde{\eta} = 0 \tag{61}$$

$$\frac{\partial \tilde{\eta}}{\partial t} + U_\infty \left(\frac{\partial \eta}{\partial x} + \frac{\partial \tilde{\eta}}{\partial x}\right) = \tilde{\varphi}_z \tag{62}$$

e, come è ben noto, le modulazioni dell'ampiezza e del numero d'onda dovute alla corrente uniforme sono semplicemente date da (vedi ad esempio Dean e Dalrymple, 1992):

$$A = A_0 \left(1 - \frac{U_\infty k}{\sigma}\right) \tag{63}$$

$$\sigma = U_\infty k + \sqrt{gk \tanh(kh)} \tag{64}$$

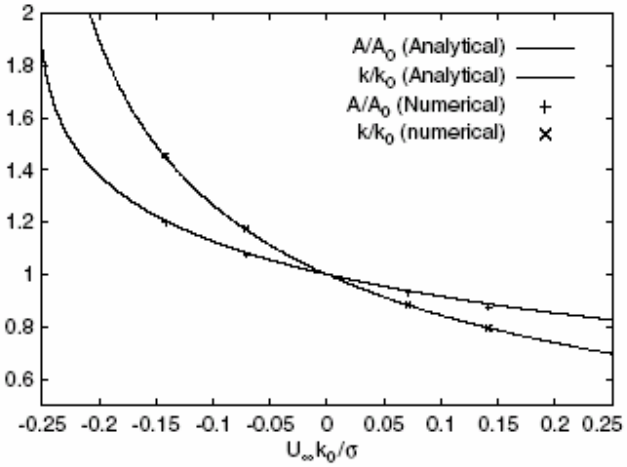

Fig. 2 Interazione tra una corrente uniforme e un'onda lineare (σ=5.54 s^{-1}, h_0=1m, k_0=πm^{-1}); sono stati utilizzati 20 elementi per lunghezza d'onda.

La Figura 2 mostra la comparazione tra la soluzione analitica e il risultato numerico ottenuto attraverso il Boundary Element Method (BEM) nella formulazione desingolarizzata.
Il lato destro delle curve delinea l'appiattimento dell'onda dovuto alla sovrapposizione dell'onda con la corrente, analogo al caso del downshifting sperimentato nel caso di strutture che determina il frangimento dell'onda.
Nella parte sinistra delle curve, invece, viene evidenziato l'aumento di ripidità dell'onda prodotta dalla corrente opposta; come principio, è possibile determinare il valore limite della corrente nel caso di corrente opposta al verso di propagazione del moto ondoso.
Tuttavia, in questo caso, la soluzione lineare diventa sempre più senza significato tanto più il modulo della velocità cresce, dal momento che gli effetti non lineari diventano sempre importanti. Inoltre, il frangimento genera una corrente opposta che consente la propagazione dell'onda solo fino a quando l'energia dell'onda non è completamente dissipata dalla rottura.

4. Bibliografia

Bassanini, P, Bulgarelli, U, Campana, E, and Lalli, F (1994). "The Wave Resistance Problem in a Boundary Integral Formulation," *Surv Math Ind*, Vol 4, pp 151–194.

Baker, GR, Meiron, DI and Orszag, SA (1989). "Generalized Vortex Methods for Free Surface Flow Problems. ii: Radiating Waves," *J Sci Comp*, Vol 4, pp 237–259.

Cao, Y, Schultz, WW and Bech, RF (1991). "Three-Dimensonal Desingularized Boundary Integral Methods for Potential Problems," *Int J Num Meth Fluids*, Vol 12, pp 785–803.

Clercx, HJH, Zoeteweij, ML, and van Heijst, GJF (2003). "The Role of Bottom Friction and Fluid Layer Depth," *Int Symp on Shallow Flows*, Delft, The Netherlands.

Dean, RG, and Dalrymple, RA (1992). *Water Wave Mechanics for Engineers and Scientists*, World Scientific, Singapore.

Guignard, S, Grilli, S, Marcer, R, and Rey, V (1999). "Computation of Shoaling and Breaking Waves in Nearshore Areas by the Coupling of BEM and VOF Methods," *Proc 9th Int Offshore*.

Hess, JL and Smith, A (1966). "Calculation of Potential Flows around Arbitrary Bodies," *Prog Aer Sci*, No 8, p 1.

Ito, Y, and Tanimoto, K (1972). "A Method of Numerical Analysis of Wave Propagation–Application to Wave Diffraction and Refraction," 13*th Int Conf Coastal Eng*, ASCE, Vol 1.

Lalli, F, Di Mascio, A, and Landrini, M (1996). "Nonlinear Diffraction Effects Around a Surface-piercing Structure," *Int J Offshore and Polar Eng*, ISOPE, Vol 6, No 2, pp 104–111.

Lalli, F (1997). "On the Accuracy of the Desingularized Boundary Integral Method in Free-Surface Flow Problems," *Int J Numer Methods in Fluids*, Vol 25, pp 1163–1184.

Lalli, F, Miozzi, M, and Romano, GP (2002). "Some Remarks on River Mouth Flow," *Proc 12th Int Offshore and Polar Eng Conf*, Kitakyushu, Japan, ISOPE, Vol 3, pp 659–664.

Landrini, M (1994). "Fenomeni non Lineari nella Propagazione di Onde di Superficie Libera," *PhD dissertation*, Dept of Mechanics and Aeronautics, Univ of Rome, La Sapienza, Rome.

Le, H, and Moin, P (1991). "An Improvement of Fractional Step Methods for the Incompressible Navier-Stokes Equations," *J Comput Phys*, Vol 92, pp 369–379.

Longuet-Higgins, MS, and Stewart, RW (1964). "Radiation Stresses in Water Waves: A Physical Discussion, with Application," *Deep-Sea Res*, Vol 11, pp 529–562.

McComb, WD (2000). *The Physics of Fluid Turbulence*, Oxford Sci Publ, Oxford.

Mei, CC (1982). *The Applied Dynamics of Ocean Surface Waves*, John Wiley & Sons, New York.

Nezu, I, and Nakagawa, H (1993). *Turbulence in Open-channel Flows*, A. A. Balkema, Rotterdam.

Spalart, PR, and Allmaras, SR (1994). "A One-equation Turbulence Model for Aerodynamic Flows," *La Recherche Aérospatiale*, No 1, pp 5–21.

Teng, B, Zhao, M, and Bai, W (2001). "Wave Diffraction in a Current over a Local Shoal," *Coastal Eng*, Vol 42, pp 163–172.

5. Appendice

A conclusione della presente tesi, nella quale sono stati analizzati e studiati i fenomeni di interazione tra il flusso di un fiume immissario ed il moto ondoso con eventuali strutture costiere, è stato implementato un codice di calcolo redatto in linguaggio FORTRAN.
Tale codice, come già evidenziato in precedenza, si basa sul Boundary Element Method per quanto concerne la definizione del moto ondoso a flusso potenziale, e sul Finite Element Method, per quanto quanto il flusso viscoso.

```
program vispo                    ! VIScous POtential flow !
-----------------------------------------------------------------------
Diffrazione di un sistema ondoso indotta da un martufo immerso
Metodo Distribuzione di Sorgenti e nuclei desingolarizzati
Integrazione nel tempo Runge-Kutta classico.
PROBLEMA LINEARIZZATO (Super Linear Unsteady Waves)

Dinamica di un getto in acque basse
Rai & Moin Finite Difference (Staggered Apat SHallow wAter model)

Wave-Jet Interaction  (Sluw - Sasha)
-----------------------------------------------------------------------
 integer*4 lat,long,npan
 parameter(lat=24,long=24,npan=lat*long)
 integer*4 nlo,ntra,nplib
 parameter(nlo=48,ntra=72,nplib=ntra*nlo)
 integer mx,my,mx1,my1,mx2,my2
 parameter(mx=49,my=73,mx1=mx-1,my1=my-1,mx2=mx-2,my2=my-2)
 real*8

Geometria Martufo
&       xm(npan),ym(npan),zm(npan),
&       x1(npan),x2(npan),x3(npan),x4(npan),xh(npan),
&       y1(npan),y2(npan),y3(npan),y4(npan),yh(npan),
&       z1(npan),z2(npan),z3(npan),z4(npan),zh(npan),
&       cosxn(npan),cosyn(npan),coszn(npan),area(npan),
$      hmax,D,

Geometria  s. l. e spiaggia
&       xn(nplib),yn(nplib),xmonte,xvalle,ylato,yta1,yta2,etata,
&       spinm,spinv,spilen,spimax,dist,spinu(nplib),smorz(nplib),

Potenziale di PERTURBAZIONE
&       fi(nplib),fin(nplib),dfi(nplib),tofi(nplib),
```

```
     &       fix(npan+nplib),fiy(npan+nplib),fiz(npan+nplib),
     &       eta(nplib),etan(nplib),deta(nplib),teta(nplib),
     &       etax(nplib),etay(nplib),

     Matrici di influenza, densita di sorgente, sistema di eq
     &       vx(nplib,npan),vy(nplib,npan),vz(nplib,npan),
     &       tx(nplib,nplib),ty(nplib,nplib),tz(nplib,nplib),
     &       txx,tyy,tzz,sigma(npan+nplib),appo(npan+nplib),
     &       a(npan+nplib,npan+nplib),rcond
      integer*4 indx(npan+nplib),iatto(npan),nnnn

    Campo Ondoso Incidente
      real*8 Am,Wn,Om,gra,Peri,Wl,fiw,uw,vw,ww,phase,eta0(nplib),
     &       eta0y(nplib),fi0y(npan+nplib),fi0z(npan+nplib)

   Runge--Kutta
      integer*4 ik,ikutta,ijk,iii
      real*8 time,trk,dt,rk(1:4,1:2)

   Collegamento Sasha/Sluw
      real*8 usa(nplib),vsa(nplib),etasa(nplib),etasax(nplib),
     $       etasay(nplib),disa(nplib),vivasa(nplib),
     $       usama(npan),vsama(npan)

   Varie Sluw
      integer*4 iter,ist,kgri,ima,i1,i2,maxite,ksta,i,j,k,idt
      real*8 zsor,zzsor,xlam,ani,pig,eps,upiep,hemips,dxl,dyl,area_sl,
     $       cfinf,xh_sl,yh_sl,diana,xnq,ynq,znq,xnt,ynt,znt,somx,
     $       somy,somz,r,rz,r2dx,r2dy
      character*20 nfile
      character*4 cha
      character*1 label
-----------------------------------------------------------------------!
                    SASHA                                              !
-----------------------------------------------------------------------!
 h     : profondità (+ verso il basso da swl)                          !
 eta   : sopraelevazione ondosa profondità (+ verso l'alto da swl)     !
 chi   : concentrazione*prof (staggered)                               !
 chiplo : concentrazione (non-stag)                                    !
 uplo  : velocità x mediata lungo la verticale (non-stag)              !
 vplo  : velocità y mediata lungo la verticale (non-stag)              !
 u     : uplo*h (staggered)                                            !
 v     : vplo*h (staggered)                                            !
 veli  : velocità litoranea                                            !
 ibos  : sponda sinistra (non stag.)                                   !
 ibod  : sponda destra (non stag.)                                     !
 jbol  : estensione y iniz                                             !
```

```
  jbol2  : estensione y vel. definitiva (< 101 )                    !
  ibos   : numero nodo bordo sinistro (foce)                          !
  ibod   : numero nodo bordo destro   (foce)                           !
  ibod-ibos < 21                                                !
  noradi : nord radiation condition                                 !
-----------------------------------------------------------------------!
      integer kpri,kru,noradi,ij,lip_w,lip_e,lip_s,istre_w,istre_e,
     $ mum,latp1,longp1,jstre,ibos,ibod,jbol,ider_w,ider_e,
     $ ider_n,lami,injet,invisco,legge,isaspi,isasha
      integer iw_eta(mx,my),iw_u(mx,my),iw_v(mx,my),iw_plo(mx,my)
      real*8 u(mx,0:my),v(0:mx,my),uv(mx,0:my),vv(0:mx,my),
     $ Hu(2,mx,my),Hv(2,mx,my),prof(mx,my),x(mx,my),y(mx,my),
     $ uplo(mx,my),vplo(mx,my),proplo(mx,my),viva(mx,my),cappa(mx,my),
     $ vor(mx,my),raprof(mx,my)
      real*8 seta(mx,my),setav(mx,my),setavv(mx,my),Hh(2,mx,my),
     $ setaplo(mx,my),saspi(mx,my)
      real*8 dx(mx),dy(my),xflow(mx),alf(3),roh(3),gam(3),ann,tualg,
     $ velmo,zer,utt,uqt,umz,utz,uno,due,tre,qua,cin,sei,die,toll,
     $ Reynolds,Foce,V0,bos,bod,bom,Q1,Q2,Q3,Q4,Bilancio,recel,cfl,
     $ dm_0,dm_1,dm_2,scamu,scamv,strex_w,strex_e,strey,rex,rey,cf,
     $ div,scau,scav,scamm,hhh,hjp,hjm,hip,him,greta,tro,vpx,vmx,vpy,
     $ vmy,ott,ove,xxx,seta0,hij,up,um,vp,vm,yyy,set,cau,cav,visco,
     $ viscolami,bol,x0,y0,smart,
     $ xappo(mx,my),yappo(mx,my),zappo(mx,my)
**************************************************************************
      zer = 0.0d0                                                !
      utt = 0.125d0                                               !
      uqt = 0.25d0                                                !
      utz = 1.d0/3.d0                                             !
      umz = 0.50d0                                                  !
      uno = 1.0d0                                                !
      due = 2.0d0                                                 !
      tre = 3.0d0                                               !
      qua = 4.0d0                                                 !
      cin = 5.0d0                                               !
      sei = 6.0d0                                               !
      set = 7.0d0                                               !
      ott = 8.d0                                               !
      ove = 9.d0                                                !
      die = 10.0d0                                                !
      gra = 9.806d0                                                 !
      pig = qua*atan(uno)                                             !
      roh(1) = zer                                              !
      gam(1) = 8.0d0/15.0d0                                             !
      roh(2) = - 17.0d0/60.0d0                                           !
      gam(2) = cin/12.0d0                                            !
      roh(3) = - gam(2)                                             !
```

```
      gam(3) = tre/qua                                          !
      do i = 1,3                                              !
        alf(i) = roh(i)+gam(i)                                  !
      end do                                                   !
      toll = .0001d0                                            !
 129  toll = umz*toll                                           !
      if((toll+uno).ne.uno) go to 129
      toll = die*toll                                           !
      smart = toll                                              !
*************************************************************************
                  LETTURA DATI DA FILE                          !
*************************************************************************
      open(unit=20,file='vispo.in',status='old')
        read(20,*)
        read(20,*) Am,Wl,D                          ! roba di sluw !
        read(20,*)
        read(20,*) idt,ikutta,maxite,ksta,zzsor
        read(20,*)
        read(20,*) diana,hmax,spinm,spinv,spilen,spimax
        read(20,*)
        read(20,*) label,kgri
        read(20,*)
        read(20,*) kpri,ider_w,ider_e,ider_n,noradi    ! roba di sasha !
        read(20,*)
        read(20,*) V0,ann,tualg,isaspi,isasha
        read(20,*)
        read(20,*) lip_w,lip_e,lip_s,istre_w,istre_e,jstre
        read(20,*)
        read(20,*) strex_w,strex_e,strey,Reynolds
        read(20,*)
        read(20,*) lami,ibos,ibod,injet,invisco,legge
      close(20)
*************************************************************************
      dxl = D/float(lat)
      dyl = D/float(long)
      r2dx = uno/(due*dxl)
      r2dy = uno/(due*dyl)
      nfile=label//'vispo.out'
      open(unit=19,file=nfile,status='unknown')
      rewind(19)
      write(19,*)
      write(19,*)' mx=',mx,'  my=',my
      write(19,*)' dx=',dxl,'  dy=',dyl
      write(19,*)
      ann = ann**2
      seta0 = zer
```

```
      Wn   = due*pig/Wl
      Peri = due*pig/sqrt(Wn*gra*tanh(hmax*Wn))
      Om   = due*pig/Peri
      dt = Peri/float(idt)
      phase = pig

      write(*,*)
      write(*,*)' CAREFUL: DIANA=',sngl(diana)
      write(*,*)

      ani = .164d0
      eps = .0001
1     eps = eps/due
      upiep = uno+eps
      if(upiep.ne.uno) go to 1
      eps = eps*due
      hemips = upiep
123   hemips = hemips/due
      if((hemips+uno).ne.uno) go to 123
      hemips = hemips*due

    COEFFICIENTI Ru-Ku  Wave
      call rkiniz (ikutta,rk)
------------------------------------------------------------------
            GEOMETRIA MARTUFO
------------------------------------------------------------------
      call martufo(nplib,lat,long,npan,xm,ym,zm,
     $     x1,x2,x3,x4,xh,y1,y2,y3,y4,yh,z1,z2,z3,z4,zh,diana,
     $     area,cosxn,cosyn,coszn,a,hmax,iatto)
------------------------------------------------------------------
            GEOMETRIA SUPERFICIE LIBERA
------------------------------------------------------------------
      xlam = Wl/dyl
      area_sl   = dxl*dyl
      zsor   = zzsor*(area_sl**( xlam))!*(Wl**(1.-2.*ani)) !!!!!!!!!!!!!!!
      write(*,*) 'nlam:',sngl(ani)
      xh_sl  = float(nlo)*dxl/due
      yh_sl  = float(ntra)*dyl/due

      k = 1
      do i = 1,nplib
        xn(i) = -xh_sl+dxl*umz+float(k-1)*dxl
        if(i.eq.k*ntra) k=k+1
      end do
      k=0
      do i=1,nplib
        yn(i) = -yh_sl+float(i-k*ntra-1)*dyl + dyl*umz
```

```
      if(i.eq.(k+1)*ntra) k=k+1
    end do
    xmonte = xn(1)/D*2.0
    xvalle = xn(nplib)/D*2.0d0
    ylato  = yn(ntra)/D*2.0d0
    write(*,*)
    write(*,*)' xmonte:',sngl(xmonte),'    xvalle:',sngl(xvalle)
    write(*,*)' ylato:',sngl(ylato)
    write(*,*)
-----------------------------------------------------------------------
    spinm = spinm*D
    spinv = spinv*D
    spilen = spilen*D
    do i=1,nplib/2
      dist = abs(yn(i))      ! sqrt(xn(i)**2+yn(i)**2)
      spinu(i) = zer
      if(dist.gt.spinm)
        if (spinu(i).gt.spimax) spinu(i) = spimax
    end do
    do i=nplib/2+1,nplib
      dist = abs(yn(i))      ! sqrt(xn(i)**2+yn(i)**2)
      spinu(i) = zer
      if(dist.gt.spinm)   !  if(dist.gt.spinv)
     $    spinu(i) = spilen *((-spinv+dist)/spilen)**2
      if (spinu(i).gt.spimax) spinu(i) = spimax
    end do
    do i = 1,nplib
      dist = abs(yn(i))
      smorz(i) = exp(-(dist/(spinm))**100)
    end do

    do i = 1,nplib
      j = 1 + (i-1)/ntra
      k = i - (j-1)*ntra
      if(yn(i).gt.zer) then       ! sistema centrato sul martufo !
        saspi(j,k) = float(isaspi)*spinu(i)
      else
        saspi(j,k) = zer
      end if
    end do
-----------------------------------------------------------------------

 Sistema ondoso incidente

    time = zer
    call Wave (npan,ym,zm,phase,
    $          nplib,yn,eta0,eta0y,fi0y,fi0z,Am,Wn,Om,hmax,gra,time)
```

```
 Campo di perturbazione

    do i=1,nplib
      eta(i)  = zer
      etan(i) = eta(i)
      fi(i)   = zer
      fin(i)  = zer
    end do
-----------------------------------------------------------------------
 OUTPUT CONDIZIONI INIZIALI
-----------------------------------------------------------------------
 Curve livello
    ist = 0
    write(cha,99) ist
    nfile= label//'liv'//cha//'.dat'
    open(99,file=nfile,form='unformatted')
        rewind(99)
        write(99) ntra,nlo,1
        write(99) 0.,0.,0.,0.
        write (99)(1.              ,i=1,nplib),
   $             (1.              ,i=1,nplib),
   $             (sngl(smorz(i))  ,i=1,nplib),
   $             (sngl(eta0(i))   ,i=1,nplib),
   $             (sngl(spinu(i))  ,i=1,nplib)
    close(99)
 Superficie libera
    nfile= label//'gri'//cha//'.dat'
    open (99,file=nfile,form='unformatted')   ! griglia plot3d !
         rewind(99)
         write(99) ntra,nlo,1
         write(99) (sngl(xn(i)),i=1,nplib),
   $              (sngl(yn(i)),i=1,nplib),
   $              (sngl(eta0(i)),i=1,nplib)
    close(99)
 Grid SL
    nfile = label//'grinu.dat'
    open (99,file=nfile,form='unformatted')   ! griglia plot3d !
    rewind(99)
    write(99) ntra,nlo,1
    write(99) (sngl(xn(i)),i=1,nplib),
   $         (sngl(yn(i)),i=1,nplib),
   $         (0.         ,i=1,nplib)
    close(99)

    cfinf = dt/dxl*Wl/Peri
    cfl =  V0*dt/dxl
```

```
      write(*,*) 'cfl_W:',sngl(cfinf),'  cfl_V:',sngl(cfl)

          write(19,*) 'Am,Wl,Peri'
          write(19,*) sngl(Am),sngl(Wl),sngl(Peri)
          write(19,*) 'idt,ikutta,maxite,ksta,xlam,zzsor'
          write(19,*) idt,ikutta,maxite,ksta,sngl(xlam),sngl(zzsor)
          write(19,*) 'diana,hmax,dxl,dyl'
          write(19,*)  sngl(diana),sngl(hmax),dxl,dyl
          write(19,*) 'spinm,spinv,spilen,spimax'
          write(19,*)  sngl(spinm),sngl(spinv),sngl(spilen),
     $                 sngl(spimax)
          write(19,*) 'label, nlo, ntra, lat, long, CFL_W, CFL_V'
          write(19,*) label,nlo,ntra,lat,long,sngl(cfinf),sngl(cfl)
          write(19,*)
          write(19,*)' xmonte:',sngl(xmonte),'    xvalle:',sngl(xvalle)
          write(19,*)' ylato:',sngl(ylato)
          write(19,*)

      nfile= label//'yeta0000'
      open(unit=20,file=nfile,status='unknown')               ! talon !
      rewind(20)
      do k = nplib/2+ntra,nplib/2+1,-1
        write(20,*) sngl(yn(k)),sngl(eta0(k)),sngl(spinu(k)),
     $                                 sngl(smorz(k))
      end do
      close(20)
      write(*,*)

-----------------------------------------------------------------------
      yta1 = - 1.6666666666667d0                    ! secondo taglio !
        ima = 0
        do i = 1,ntra
          if(yn(i).ge.yta1.and.ima.eq.0) then
            i1 = i
            ima = 1
          end if
        end do
      yta2 = - 2.4666666666667d0                      ! terzo taglio !
        ima = 0
        do i = 1,ntra
          if(yn(i).ge.yta2.and.ima.eq.0) then
            i2 = i
            ima = 1
          end if
        end do
-----------------------------------------------------------------------
           GEOMETRIA PER SASHA                                !
```

```
------------------------------------------------------------------

      nfile = 'martufo.sas'
      open (99,file=nfile,form='unformatted',status='old')
      rewind(99)
      read(99) latp1,longp1,mum
      read(99)((xappo(i,j),i=1,latp1),j=1,longp1),
     $        ((yappo(i,j),i=1,latp1),j=1,longp1),
     $        ((zappo(i,j),i=1,latp1),j=1,longp1)
      close(99)
      if(latp1.ne.(lat+1)) then
        write(*,*)'incongruenza lat carshoal/vispo'
        stop
      end if
      if(longp1.ne.(long+1)) then
        write(*,*)'incongruenza long carshoal/vispo'
        stop
      end if

      if(nlo.ne.(2*lat)) then
        write(*,*) 'careful grid lat/nlo'
        stop
      end if
      if(ntra.ne.(3*long)) then
        write(*,*) 'careful grid long/ntra'
        stop
      end if
      do i = 1,lat+1
        do j = 1,long+1
          x(i+lat/2,j+long) = xappo(i,j)
          y(i+lat/2,j+long) = yappo(i,j)
          proplo(i+lat/2,j+long) = zappo(i,j)
        end do
      end do
      dx(1) = abs(x(1+lat/2,1+long)-x(2+lat/2,1+long))
      do i = 2,mx1
        dx(i) = dx(1)
      end do
      dy(1) = abs(y(1+lat/2,1+long)-y(1+lat/2,2+long))
      do i = 2,my1
        dy(i) = dy(1)
      end do
      do i = lat/2+1,3*lat/2+1
        do j = 1,long
          x(i,j) = x(i,long+1)
          proplo(i,j) = hmax
        end do
```

```
      end do
      do i = lat/2+1,3*lat/2+1
        do j = 2*long+2,my
          x(i,j) = x(i,long+1)
          proplo(i,j) = hmax
        end do
      end do
      do i = lat/2,1,-1
        do j = 1,my
          x(i,j) = x(i+1,j) - dx(i)
          proplo(i,j) = hmax
        end do
      end do
      do i = 3*lat/2+1,mx1
        do j = 1,my
          x(i+1,j) = x(i,j) + dx(i)
          proplo(i+1,j) = hmax
        end do
      end do
      do i = 1,lat/2
        do j = long+1,2*long+1
          y(i,j) = y(lat/2+1,j)
        end do
      end do
      do i = 3*lat/2+2,nlo+1
        do j = long+1,2*long+1
          y(i,j) = y(lat/2+1,j)
        end do
      end do
      do i = 1,mx
        do j = long,1,-1
          y(i,j) = y(i,j+1) - dy(j)
        end do
      end do
      do i = 1,mx
        do j = 2*long+1,my1
          y(i,j+1) = y(i,j) + dy(j)
        end do
      end do
      do i = 1,mx1
        do j = 1,my1
          prof(i,j) = uqt*(proplo(i,j)+proplo(i+1,j)+proplo(i,j+1)+
     $                proplo(i+1,j+1))
        end do
      end do

      open(18,file='geo.dat',form='unformatted')
```

```
      rewind(18)
      write(18) mx,my
      write(18) ((sngl(x(i,j)),i=1,mx),j=1,my),
     $          ((sngl(y(i,j)),i=1,mx),j=1,my)
      close(18)
      open(18,file='geosta.dat',form='unformatted')
      rewind(18)
      write(18) mx1,my1
      write(18) ((sngl(x(i,j)+umz*dx(i)),i=1,mx1),j=1,my1),
     $          ((sngl(y(i,j)+umz*dy(j)),i=1,mx1),j=1,my1)
      close(18)

      jbol = lat/2 + 1
      x0 = zer
      y0 = -yh_sl
      bos = x(1,1) + float(ibos-1)*dx(1)
      bod = x(1,1) + float(ibod-1)*dx(1)
      write(*,*)
      write(*,*) ' prova bos (must be 0)',sngl(bos-(x(ibos,1)))
      write(*,*) ' prova bod (must be 0)',sngl(bod-(x(ibod,1)))
      write(*,*)

      Foce = bod-bos
      bom = (bod+bos)/due
      bol = float(jbol-1)*dy(1)

      do j = 1,my
        do i = 1,mx
          iw_plo(i,j) = uno
        end do
      end do
      do j = 1,my1
        do i = 1,mx1
          iw_eta(i,j) = uno
        end do
      end do
      do j = 1,my1
        do i = 1,mx
          iw_u(i,j) = uno
        end do
      end do
      do j = 1,my
        do i = 1,mx1
          iw_v(i,j) = uno
        end do
      end do
      do j = 1,my1
```

```
      do i = 1,mx1
        iw_eta(i,j) = uno
      end do
    end do
    do i = ibos-2,ibos                    ! molo guardiano sin !
      do j = 1,jbol
        iw_plo(i,j) = zer
      end do
    end do
    do i = ibos-2,ibos
      do j = 1,jbol-1
        iw_u(i,j) = zer
      end do
    end do
    do i = ibos-2,ibos-1
      do j = 1,jbol
        iw_v(i,j) = zer
      end do
    end do
    do i = ibos-2,ibos-1
      do j = 1,jbol-1
        iw_eta(i,j) = zer
      end do
    end do
    do i = ibod,ibod+2                     ! molo guardiano dx !
      do j = 1,jbol
        iw_plo(i,j) = uno
      end do
    end do
    do i = ibod,ibod+2
      do j = 1,jbol-1
        iw_u(i,j) = uno
      end do
    end do
    do i = ibod,ibod+1
      do j = 1,jbol
        iw_v(i,j) = uno
      end do
    end do
    do i = ibod,ibod+1
      do j = 1,jbol-1
        iw_eta(i,j) = zer
      end do
    end do
.....................................................................
    write(*,*)' bos:',sngl(bos),'bod:',sngl(bod),'bol:',sngl(bol)
    write(*,*)' Foce:',sngl(Foce),' hmax:',sngl(hmax)
```

```
      write(*,*)' x0:',sngl(x0),'  bom:',sngl(bom),'  y0:',sngl(y0)
      write(19,*)' bos:',sngl(bos),' bod:',sngl(bod),' bol:',sngl(bol)
      write(19,*)' Foce:',sngl(Foce),' hmax:',sngl(hmax)
      write(19,*)' x0:',sngl(x0),'  bom:',sngl(bom),'  y0:',sngl(y0)
.....................................................................

..................... flusso iniziale .....................!
      do i = 1,mx
        do j = 1,my
          uplo(i,j) = zer
          vplo(i,j) = zer
          seta(i,j) = zer
          setav(i,j) = seta(i,j)
          setavv(i,j) = seta(i,j)
        end do
      end do

      do i = 1,mx
        xflow(i) = uno - ((x(i,1)-bom)/(bod-bom))**4
        if(xflow(i).lt.zer) xflow(i) = zer
        do j = 1,my
          vplo(i,j) = V0*xflow(i)*(uno-((y(1,j)-y0)/(1.5*bol))**8)
        if(vplo(i,j).lt.zer) vplo(i,j) = zer
        end do
      end do

.....................................................................

      do i = 1,mx
        u(i,0) = zer
        uv(i,0) = u(i,0)
        do j = 1,my1
          u(i,j) = uqt*(uplo(i,j)+uplo(i,j+1))*
     $               (proplo(i,j)+proplo(i,j+1))
          uv(i,j) = u(i,j)
        end do
        u(i,my) = zer
        uv(i,my) = u(i,my)
      end do

      do j = 1,my
        v(0,j) = zer
        vv(0,j) = zer
        do i = 1,mx1
          v(i,j) = uqt*(vplo(i,j)+vplo(i+1,j))*
     $               (proplo(i,j)+proplo(i+1,j))
          vv(i,j) = v(i,j)
```

```
      end do
      v(mx,j) = zer
      vv(mx,j) = zer
    end do
    Q1 = zer
    do i = 1,mx1
      Q1 = Q1 + v(i,1)*dx(i)
    end do
    viscolami = Q1/(hmax*Reynolds)
    write(*,*)''
    write(*,*)'viscolami=',viscolami
    write(*,*)''
    if(lami.eq.0) then
      visco = .000001d0
      Reynolds = Q1/(hmax*visco)
    else if(lami.eq.1) then
      visco = Q1/(hmax*Reynolds)
    end if

.....................................................................

    nfile=label//'vispo.sto'
    open(unit=44,file=nfile,status='unknown')
    rewind(44)
    write(*,779) mx,my,Reynolds,dt,dx(1),dy(1)
    write(19,779) mx,my,Reynolds,dt,dx(1),dy(1)
779  format(/,' ************ Wave - Jet Interaction ******* ',//,
   $ '  mx =',i3,' my =',i3,'  Reynolds =',g9.2/
   $ '  dt =',f7.4,'  dx(1) =',f12.4,'  dy(1) =',f12.4,//)

    do i = 1,mx
      do j = 1,my
        Hh(1,i,j) = zer
        Hh(2,i,j) = zer
        Hu(1,i,j) = zer
        Hu(2,i,j) = zer
        Hv(1,i,j) = zer
        Hv(2,i,j) = zer
      end do
    end do

    nfile=label//'inx.dat'
    open(unit=66,file=nfile,status='unknown')
    rewind(66)
      do i = 1,mx1
        write(66,222) sngl(x(i,1)),sngl(umz*(x(i,1)+x(i+1,1))),
   $ sngl(vplo(i,1)),sngl(v(i,1)),iw_eta(i,1),iw_u(i,1),iw_v(i,1),
```

```
     $ iw_plo(i,1),xflow(i),proplo(i,1),prof(i,1)
       end do
222     format(4(1x,f12.4),4(1x,i4),3(1x,f12.4))
      close(66)
      nfile=label//'iny.dat'
      open(59,file=nfile,status='unknown')
      rewind(59)
        i = 1+istre_w+(ibos+ibod)/2
        do j = 1,my
          write(59,*) sngl(y(1,j)),
     $                sngl(vplo(i,j)),
     $               sngl(proplo(i,j))
        end do
      close(59)
      nfile=label//'sa0000.dat'
      open(59,file=nfile,form='unformatted')
      rewind(59)
      write(59) mx,my
      write(59) 1., 1., sngl(Reynolds), sngl(time)
      write(59)
     %    ((1.,i=1,mx),j=1,my),
     %    ((sngl(uplo(i,j)),i=1,mx),j=1,my),
     %    ((sngl(vplo(i,j)),i=1,mx),j=1,my),
     %    ((sngl(proplo(i,j)),i=1,mx),j=1,my)
      close(59)
      nfile=label//'iw_plo.dat'
      open(59,file=nfile,form='unformatted')
      rewind(59)
      write(59) mx,my
      write(59) 1., 1., sngl(Reynolds), sngl(time)
      write(59)
     %    ((1.,i=1,mx),j=1,my),
     %    ((iw_u(i,j),i=1,mx),j=1,my),
     %    ((iw_v(i,j),i=1,mx),j=1,my),
     %    ((iw_plo(i,j),i=1,mx),j=1,my)
      close(59)
      write(19,*)' portata:', sngl(Q1),
     $ ' m3/sec;   F_h:',sngl(V0/sqrt(gra*hmax))
      write(*,*)' portata:', sngl(Q1),
     $ ' m3/sec;   F_h:',sngl(V0/sqrt(gra*hmax))

.....!

      do i = 1,mx1
        do j = 1,my1
          raprof(i,j) = sqrt(ann*gra/(prof(i,j)**utz))
        end do
```

```
      end do


      if(kgri.eq.1) stop
------------------------------------------------------------------
 COEFFICIENTI INFLUENZA
------------------------------------------------------------------

 Influenza Superficie Libera

      if(injet.eq.0) then
.....*
      write(*,*)'---------------- SUPERFICIE LIBERA ========--------'
      write(*,*)'---------------- S.L. --> MARTUFO  ========--------'
      do 30 k=1,nplib   ! -- punto q --
        xnq = xn(k)
        ynq = yn(k)
        znq = zsor
       Superficie Libera ==> Martufo
        do 40 i=1,npan   ! -- punto p del martufo --
          xnt=xm(i)
          ynt=ym(i)
          znt=zm(i)
          r  = sqrt((xnq - xnt)**2+( ynq - ynt)**2+(znt-znq)**2)
          rz = sqrt((xnt-xnq)**2+(ynt-ynq)**2+(znt+(znq+2.*hmax))**2)

          txx  = -(xnt-xnq)*(uno/r**3 + uno/rz**3)
          tyy  = -(ynt-ynq)*(uno/r**3 + uno/rz**3)
          tzz = -(znt-znq)/r**3-(znt+(znq+2.*hmax))/rz**3

          a(i,k+npan) = txx*cosxn(i) + tyy*cosyn(i) + tzz*coszn(i)

 40    continue
       Superficie Libera ==> Superficie Libera
        do 45 i=1,nplib   ! -- punto p della s. l. --
          xnt=xn(i)
          ynt=yn(i)
          znt=0.0
          r  = sqrt((xnt-xnq)**2+(ynt-ynq)**2+(znt-znq)**2)
          rz = sqrt((xnt-xnq)**2+(ynt-ynq)**2+(znt+(znq+due*hmax))**2)
          tx(i,k) = -(xnt-xnq)*(uno/r**3+uno/rz**3)
          ty(i,k) = -(ynt-ynq)*(uno/r**3+uno/rz**3)
          tz(i,k) = -(znt-znq)/r**3-(znt+(znq+2.*hmax))/rz**3
          a(npan+i,npan+k) = uno/r + uno/rz
 45    continue
 30   continue
```

```
 Influenza MARTUFO

     write(*,*) '--------=========  MARTUFO --> S.L. ==========----------'

     call caldic(x1,x2,x3,x4,xh,xm,cosxn,xn,
    $            y1,y2,y3,y4,yh,ym,cosyn,yn,
    $            z1,z2,z3,z4,zh,zm,coszn,eta,diana,
    $            area,a,vx,vy,vz,npan,nplib,
    $            npan+1,npan+nplib,1,npan,hmax,hemips)
-----------------------------------------------------------------------
 SOLUZIONE SISTEMA EQUAZIONI
-----------------------------------------------------------------------

     write(*,*) '---------====  FATTORIZZAZIONE MATRICE  ====----------'

     nnnn = npan + nplib
     call dgeco(a,nnnn,nnnn,indx,rcond,appo)
     write (*,*)  'Prc. di macch.  =',eps
     write (*,*)  'Rec. num. cond. =',rcond
     write (19,*) 'Prc. di macch.  =',eps
     write (19,*) 'Rec. num. cond. =',rcond
     close (19)
.....................................................................*
     end if                          ! injet !
.....................................................................*

     if(legge.eq.1) then
     open(unit=94,file='contwr.dat',status='old',form='unformatted')
     rewind(94)
       read(94) ((uplo(i,j),i=1,mx),j=1,my),
    $           ((vplo(i,j),i=1,mx),j=1,my),
    $           ((setaplo(i,j),i=1,mx),j=1,my),
    $           ((u(i,j),i=1,mx),j=0,my),
    $           ((v(i,j),i=0,mx),j=1,my),
    $           ((seta(i,j),i=1,mx1),j=1,my1),
    $           ((viva(i,j),i=1,mx1),j=1,my1)
     close(94)
     do i = 1,mx
       do j = 0,my
         uv(i,j) = u(i,j)
       end do
     end do
     do i = 0,mx
       do j = 1,my
         vv(i,j) = v(i,j)
       end do
     end do
```

```
      do i = 1,mx1
        do j = 1,my1
          setav(i,j) = seta(i,j)
        end do
      end do
      write(*,*)
      write(*,*) ' Ho letto contwr.dat (jet inizializzato)'
      write(*,*)
      end if
.....*

..... PASSAGGI SASHA/SLUW .....*
      do ijk = 1,(1-injet)
      do i = 1,nplib
        j = 1 + (i-1)/ntra
        k = i - (j-1)*ntra
        usa(i) = smorz(i)*umz*(u(j,k)+u(j+1,k))/prof(j,k)
        vsa(i) = smorz(i)*umz*(v(j,k)+v(j,k+1))/prof(j,k)
        disa(i) = (u(j+1,k)-u(j,k))/dx(j) + (v(j,k+1)-v(j,k))/dy(k)
     $         + ((u(j+1,k)-u(j,k))/dx(j) + (v(j,k+1)-v(j,k))/dy(k))*
     $           seta(j,k)/prof(j,k)
        disa(i) = smorz(i)*disa(i)
        etasa(i) = smorz(i)*seta(j,k)
        vivasa(i) = viva(j,k)
      end do
      do i = 1,npan
        k = lat + 1 + (i-1)/lat                    ! careful: lat=long !
        j = lat/2 + i - (k-lat-1)*lat
        usama(i) = umz*(u(j,k)+u(j+1,k))/prof(j,k)
        vsama(i) = umz*(v(j,k)+v(j,k+1))/prof(j,k)
      end do
      end do

=======================================================================
=======================================================================
      write(*,*) '--------====  INIZIO CICLO TEMPORALE   ====--------'
=======================================================================
=======================================================================
      do 777 iter = 1,maxite
      time = (iter-1)*dt

.....*
.....*
.....*
***********************************************************************
      do iii = 1,isasha                    ! if(isasha.eq.1) sasha va !
***********************************************************************
```

```
.....*
      do 227 kru = 1,3                        ! substeps Ru-Ku SASHA !

.....attrito sul fondo e viva.....*

      call vorti(vor,u,v,proplo,dx,dy,zer,uqt,umz,tre,ott,ove,mx,my,
     $           mx1,my1,mx2,my2,lip_w,lip_e,lip_s)
      do j = 1,my1
        do i = 1,mx1
          xxx = umz*(u(i,j)+u(i+1,j))
          yyy = umz*(v(i,j)+v(i,j+1))
          velmo = sqrt(xxx**2+yyy**2)
          cappa(i,j) = gra*ann*velmo                  ! velmo = |U| h !
          viva(i,j) = visco + float(1-lami)*tualg*raprof(i,j)*velmo
     $      + float(invisco)*float(1-lami)*viscolami*
     $            (uno-float(iter)/(float(umz)+.0001d0))
        end do
      end do
.....*


..... termini non lineari eta, u, v .....*
      do j = 1,my1                                  ! eta !
        do i = 1,mx1
          Hh(1,i,j) = - (u(i+1,j)-u(i,j))/dx(i)
     $                - (v(i,j+1)-v(i,j))/dy(j)
     $                - saspi(i,j)*seta(i,j)       ! spiaggia numerica !
        end do
      end do

      call unoli(Hu,u,v,dx,dy,prof,proplo,smart,mx,mx1,mx2,my,     ! u !
     $           my1,my2,istre_w,istre_e,jstre,ider_w,ider_e,ider_n)

-----------------------------------------------------------------------
      do ij = 1,noradi                              ! coco nord u !
        j = my
        do i = 2,mx1
          hjp = dx(i-1)**2/proplo(i+1,j)
          hij = (dx(i)**2-dx(i-1)**2)/proplo(i,j)
          hjm = dx(i)**2/proplo(i-1,j)
          xxx = dx(i)*dx(i-1)*(dx(i)+dx(i-1))
          hip = uno/proplo(i,j)
          him = uno/proplo(i,j-1)
          up = u(i,j)
          um = (dy(j-2)*u(i,j-1)+dy(j-1)*u(i,j-2))/(dy(j-1)+dy(j-2))
          vp = (dx(i-1)*v(i,j)+dx(i)*v(i-1,j))/(dx(i)+dx(i-1))
          if(vp.lt.zer) then
```

```
          tro=zer
        else if(vp.ge.zer) then
          tro=uno
        end if
        vm = (dx(i-1)*v(i,j-1)+dx(i)*v(i-1,j-1))/(dx(i)+dx(i-1))
        Hu(1,i,j) =
     $     - (hjp*u(i+1,j)**2+hij*u(i,j)**2-hjm*u(i-1,j)**2)/xxx
     $     - tro*(up*vp*hip-um*vm*him)/dy(j-1)
      end do
     end do
-----------------------------------------------------------------------
     do j = 1,my1                    ! u (correzione divergenza) !
       do i = 2,mx1
        Hu(1,i,j) = Hu(1,i,j) + (
     $     dx(i-1)*((u(i+1,j)-u(i,j))/dx(i)+(v(i,j+1)-v(i,j))/dy(j))
     $ +dx(i)*((u(i,j)-u(i-1,j))/dx(i-1)+(v(i-1,j+1)-v(i-1,j))/dy(j))
     $     )/(dx(i)+dx(i-1))*u(i,j)/(umz*(proplo(i,j)+proplo(i,j+1)))
       end do
     end do
-----------------------------------------------------------------------
     call vnoli(Hv,u,v,dx,dy,prof,proplo,smart,mx,mx1,mx2,my,     ! v !
     $          my1,my2,istre_w,istre_e,jstre,ider_w,ider_e,ider_n)
-----------------------------------------------------------------------
     do ij = 1,noradi                          ! coco nord v !
       j = my
       do i = 1,mx1
        hip = due/(proplo(i,j)+proplo(i+1,j))
        him = due/(proplo(i,j-1)+proplo(i+1,j-1))
        hjp = uno/proplo(i+1,j)
        hjm = uno/proplo(i,j)
        up = u(i+1,j)
        um = u(i,j)
        if(i.eq.1) then
          vp = (dx(i)*v(i+1,j)+dx(i+1)*v(i,j))/(dx(i)+dx(i+1))
          vm = v(i-1,j)
        else if(i.eq.mx1) then
          vp = v(i+1,j)
          vm = (dx(i)*v(i-1,j)+dx(i-1)*v(i,j))/(dx(i)+dx(i-1))
        else
          vp = (dx(i)*v(i+1,j)+dx(i+1)*v(i,j))/(dx(i)+dx(i+1))
          vm = (dx(i)*v(i-1,j)+dx(i-1)*v(i,j))/(dx(i)+dx(i-1))
        end if
        if(v(i,j).lt.zer) then
          tro=zer
        else if(v(i,j).ge.zer) then
          tro=uno
        end if
```

```
        Hv(1,i,j) =
     $     - tro*(v(i,j)**2*hip-v(i,j-1)**2*him)/dy(j-1)
     $     - (up*vp*hjp-um*vm*hjm)/dx(i)
       end do
      end do
-----------------------------------------------------------------------
      do j = 2,my1                    ! v (correzione divergenza) !
        do i = 1,mx1
          Hv(1,i,j) = Hv(1,i,j) + (
     $     dy(j-1)*((u(i+1,j)-u(i,j))/dx(i)+(v(i,j+1)-v(i,j))/dy(j))
     $    +dy(j)*((u(i+1,j)-u(i,j))/dx(i)+(v(i,j)-v(i,j-1))/dy(j-1))
     $     )/(dy(j)+dy(j-1))*v(i,j)/(umz*(proplo(i,j)+proplo(i+1,j)))
       end do
      end do
-----------------------------------------------------------------------

..... termini lineari u,v .....*
      do i = 2,mx1                              ! componente u !

        do j=1,my1
          vpx = viva(i,j)*prof(i,j)
          vmx = viva(i-1,j)*prof(i-1,j)
          hip = umz*(proplo(i+1,j)+proplo(i+1,j+1))
          hij = umz*(proplo(i,j)+proplo(i,j+1))
          him = umz*(proplo(i-1,j)+proplo(i-1,j+1))
          up = vpx*(u(i+1,j)/ hij -u(i,j)/hij)/dx(i)
          um = vmx*(u(i,j)/hij-u(i-1,j)/him)/dx(i-1)
          if(j.eq.1) then                   ! b. inf (dy(1)=dy(2)) !
            vpy = (dx(i-1)*dy(j+1)*viva(i,j)+
     $           dx(i-1)*dy(j)*viva(i,j+1)+
     $           dx(i)*dy(j+1)*viva(i-1,j)+
     $           dx(i)*dy(j)*viva(i-1,j+1))*proplo(i,j+1)/
     $          ((dx(i)+dx(i-1))*(dy(j)+dy(j+1)))
            vmy = (dx(i-1)*viva(i,j)+dx(i)*viva(i-1,j))*proplo(i,j)/
     $           (dx(i)+dx(i-1))
            hjp = umz*(proplo(i,j+1)+proplo(i,j+2))
            hjm = proplo(i,j)
            Hu(1,i,j) = Hu(1,i,j) + (
     $          (vpy*(u(i,j+1)/hjp-u(i,j)/hij)
     $          -vmy*float(lip_s)*
     $         (-ott*u(i,j-1)/hjm+ove*u(i,j)/hij-u(i,j+1)/hjp)/tre)
     $                 /dy(j)**2 )
          else if(j.eq.my1) then            ! b. sup (dy(my2)=dy(my1)) !
            vpy = (dx(i-1)*viva(i,j)+dx(i)*viva(i-1,j))*proplo(i,j+1)/
     $           (dx(i)+dx(i-1))
            vmy = (dx(i-1)*dy(j-1)*viva(i,j)+
     $           dx(i-1)*dy(j)*viva(i,j-1)+
```

```
     $            dx(i)*dy(j-1)*viva(i-1,j)+
     $            dx(i)*dy(j)*viva(i-1,j-1))*proplo(i,j)/
     $          ((dx(i)+dx(i-1))*(dy(j)+dy(j-1)))
          hjp = proplo(i,j+1)
          hjm = umz*(proplo(i,j)+proplo(i,j-1))
          Hu(1,i,j) = Hu(1,i,j) + (
     $       (vpy*(ott*u(i,j+1)/hjp-ove*u(i,j)/hij+u(i,j-1)/hjm)/tre
     $                -vmy*(u(i,j)/hij-u(i,j-1)/hjm))/dy(j)**2
     $                        )
        else
          vpy = (dx(i-1)*dy(j+1)*viva(i,j)+
     $            dx(i-1)*dy(j)*viva(i,j+1)+
     $            dx(i)*dy(j+1)*viva(i-1,j)+
     $            dx(i)*dy(j)*viva(i-1,j+1))*proplo(i,j+1)/
     $          ((dx(i)+dx(i-1))*(dy(j)+dy(j+1)))
          vmy = (dx(i-1)*dy(j-1)*viva(i,j)+
     $            dx(i-1)*dy(j)*viva(i,j-1)+
     $            dx(i)*dy(j-1)*viva(i-1,j)+
     $            dx(i)*dy(j)*viva(i-1,j-1))*proplo(i,j)/
     $          ((dx(i)+dx(i-1))*(dy(j)+dy(j-1)))
          hjp = umz*(proplo(i,j+1)+proplo(i,j+2))
          hjm = umz*(proplo(i,j)+proplo(i,j-1))
          Hu(1,i,j) = Hu(1,i,j) +      (
     $          (vpy*(u(i,j+1)/hjp-u(i,j)/hij)*due/(dy(j+1)+dy(j))-
     $           vmy*(u(i,j)/hij-u(i,j-1)/hjm)*due/
     $           (dy(j-1)+dy(j)))/dy(j)  )
        end if
        Hu(1,i,j) = Hu(1,i,j) + (
     $            due*(up-um)*due/(dx(i)+dx(i-1))
     $    + ( vpy*(v(i,j+1)/(umz*(proplo(i,j+1)+proplo(i+1,j+1)))
     $          -v(i-1,j+1)/( up *(proplo(i,j+1)+proplo(i-1,j+1))))-
     $        vmy*(v(i,j)/(umz*(proplo(i,j)+proplo(i+1,j)))
     $          -v(i-1,j)/(umz*(proplo(i,j)+proplo(i-1,j)))) )
     $             /(dy(j)* up *(dx(i)+dx(i-1)))
     $                        )
       end do
      end do

------------------------------------------------------------------------
      do ij = 1,noradi                          ! coco nord u !
        do i = 2,mx1
          j = my
          vpx = viva(i,j-1)
          vmx = viva(i-1,j-1)
          up = vpx*(u(i+1,j)-u(i,j))/dx(i)
          um = vmx*(u(i,j)-u(i-1,j))/dx(i-1)
          Hu(1,i,j) = Hu(1,i,j) + (due*(up-um)*due/(dx(i)+dx(i-1)))
```

```
      end do
    end do
.....................................................
    do j = 2,my1                              ! componente v !
      do i = 1,mx1
        vpy = viva(i,j)*prof(i,j)
        vmy = viva(i,j-1)*prof(i,j-1)
        hjp = umz*(proplo(i,j+1)+proplo(i+1,j+1))
        hij = umz*(proplo(i,j)+proplo(i+1,j))
        hjm = umz*(proplo(i,j-1)+proplo(i+1,j-1))
        vp = vpy*(v(i,j+1)/hjp-v(i,j)/hij)/dy(j)
        vm = vmy*(v(i,j)/hij-v(i,j-1)/hjm)/dy(j-1)
        if(i.eq.1) then                       ! b. sin (dx(1)=dx(2)) !
          vpx = (dx(i+1)*dy(j-1)*viva(i,j)+
     $           dx(i+1)*dy(j)*viva(i,j-1)+
     $           dx(i)*dy(j-1)*viva(i+1,j)+
     $           dx(i)*dy(j)*viva(i+1,j-1))*proplo(i+1,j)/
     $         ((dx(i)+dx(i+1))*(dy(j)+dy(j-1)))
          vmx = (dy(j-1)*viva(i,j)+dy(j)*viva(i,j-1))*proplo(i,j)/
     $           (dy(j)+dy(j-1))
          hip = umz*(proplo(i+1,j)+proplo(i+2,j))
          him = proplo(i,j)
          Hv(1,i,j) = Hv(1,i,j) + (
     $    (vpx*(v(i+1,j)/hip-v(i,j)/hij)-float(lip_w)* ! lip_w=0 f. s. !
     $     vmx*(-ott*v(i-1,j)/him+ove*v(i,j)/hij-v(i+1,j)/hip)/tre)
     $           /dx(i)**2       )
        else if(i.eq.mx1) then            ! b. dx (dx(mx1)=dx(mx2)) !
          vpx = (dy(j-1)*viva(i,j)+dy(j)*viva(i,j-1))*proplo(i+1,j)/
     $           (dy(j)+dy(j-1))
          vmx = (dx(i-1)*dy(j-1)*viva(i,j)+
     $           dx(i-1)*dy(j)*viva(i,j-1)+
     $           dx(i)*dy(j-1)*viva(i-1,j)+
     $           dx(i)*dy(j)*viva(i-1,j-1))*proplo(i,j)/
     $         ((dx(i)+dx(i-1))*(dy(j)+dy(j-1)))
          hip = proplo(i+1,j)
          him = umz*(proplo(i,j)+proplo(i-1,j))
          Hv(1,i,j) = Hv(1,i,j) + (
     $           (float(lip_e)*           ! lip_e=0 free slip est !
     $     vpx*(ott*v(i+1,j)/hip-ove*v(i,j)/hij+v(i-1,j)/him)/tre
     $    -vmx*(v(i,j)/hij-v(i-1,j)/him))/dx(i)**2
     $                        )
        else
          vpx = (dx(i+1)*dy(j-1)*viva(i,j)+
     $           dx(i+1)*dy(j)*viva(i,j-1)+
     $           dx(i)*dy(j-1)*viva(i+1,j)+
     $           dx(i)*dy(j)*viva(i+1,j-1))*proplo(i+1,j)/
     $         ((dx(i)+dx(i+1))*(dy(j)+dy(j-1)))
```

```
        vmx = (dx(i-1)*dy(j-1)*viva(i,j)+
     $         dx(i-1)*dy(j)*viva(i,j-1)+
     $         dx(i)*dy(j-1)*viva(i-1,j)+
     $         dx(i)*dy(j)*viva(i-1,j-1))*proplo(i,j)/
     $        ((dx(i)+dx(i-1))*(dy(j)+dy(j-1)))
        hip = up*(proplo(i+1,j)+proplo(i+2,j))
        him = umz*(proplo(i,j)+proplo(i-1,j))
        Hv(1,i,j) = Hv(1,i,j) +   ( (                !! careful !!
     $ float(iw_plo(i+1,j))*vpx*(v(i+1,j)-v(i,j))*due/(dx(i+1)+dx(i))-
     $  float(iw_plo(i,j))*vmx*(v(i,j)-v(i-1,j))*due/(dx(i-1)+dx(i)))
     $   vpx*(v(i+1,j)/hip-v(i,j)/hij)*due/(dx(i+1)+dx(i))-
     $   vmx*(v(i,j)/hij-v(i-1,j)/him)*due/(dx(i-1)+dx(i)))
     $                  /dx(i)  )
       end if
       Hv(1,i,j) = Hv(1,i,j) +          (
     $           due*(vp-vm)*due/(dy(j)+dy(j-1))
     $    + (vpx*(u(i+1,j)/(umz*(proplo(i+1,j)+proplo(i+1,j+1)))
     $         -u(i+1,j-1)/(umz*(proplo(i+1,j)+proplo(i+1,j-1))))
     $        -vmx*(u(i,j)/(umz*(proplo(i,j)+proplo(i,j+1)))
     $          -u(i,j-1)/(umz*(proplo(i,j)+proplo(i,j-1))) ))
     $      /(dx(i)*umz*(dy(j)+dy(j-1)))   )
     end do
    end do
.....*
    do ij = 1,noradi                          ! coco nord v !
     j = my
     i = 1                                   ! b. sin !
     vpx = umz*(viva(i,j-1)+viva(i+1,j-1))
     vmx = viva(i,j-1)
     Hv(1,i,j) = Hv(1,i,j) + due*(
     $              vpx*(v(i+1,j)-v(i,j))-
     $    float(lip_w)*vmx*(-ott*v(i-1,j)+ove*v(i,j)-v(i+1,j))/tre
     $                        )/(dx(i)**2)

     do i = 2,mx2                            ! parte centrale !
      vpx = (dx(i+1)*viva(i,j-1)+
     $        dx(i)*viva(i+1,j-1))/(dx(i)+dx(i+1))
      vmx = (dx(i-1)*viva(i,j-1)+
     $        dx(i)*viva(i-1,j-1))/(dx(i)+dx(i-1))
      Hv(1,i,j) = Hv(1,i,j) +
     $      due*(vpx*(v(i+1,j)-v(i,j))*due/(dx(i+1)+dx(i))-
     $          vmx*(v(i,j)-v(i-1,j))*due/(dx(i-1)+dx(i)))/(dx(i))
     end do

     i = mx1                                  ! b. dx !
     vpx = viva(i,j-1)
     vmx = umz*(viva(i,j-1)+viva(i-1,j-1))
```

```
      Hv(1,i,j) = Hv(1,i,j) + due*(
     $     float(lip_e)*vpx*(ott*v(i+1,j)-ove*v(i,j)+v(i-1,j))/tre
     $              -vmx*(v(i,j)-v(i-1,j))
     $                          )/(dx(i)**2)

      end do
-----------------------------------------------------------------------

..... aggiorno eta .....*

      do j = 1,my1
        do i = 1,mx1
          seta(i,j) = seta(i,j) + dt*(gam(kru)*Hh(1,i,j)+
     $                          roh(kru)*Hh(2,i,j))
          Hh(2,i,j) = Hh(1,i,j)              ! aggiorno termine noto !
        end do
      end do


..... aggiorno u .....*

      do i = 2,mx1
        do j = 1,my1
          hhh = (dx(i-1)*prof(i,j)+dx(i)*prof(i-1,j))/(dx(i-1)+dx(i))
          greta = gra*hhh*(seta(i,j)-seta(i-1,j))*due/(dx(i-1)+dx(i))
          hhh = hhh*(  (dx(i-1)*prof(i,j)+dx(i)*prof(i-1,j))/
     $               (dx(i-1)+dx(i))  )**(qua/tre)
         hhh = hhh**(set/tre)
          cau = (dx(i-1)*cappa(i,j)+dx(i)*cappa(i-1,j))/(dx(i)+dx(i-1))
          Hu(1,i,j) = Hu(1,i,j) - saspi(i,j)*u(i,j)          ! spi num !
          u(i,j) = float(iw_u(i,j))*
     $           hhh*( u(i,j) + dt*(gam(kru)*Hu(1,i,j)+
     $                          roh(kru)*Hu(2,i,j)
     $                         -alf(kru)*greta) )/
     $                       (hhh+dt*alf(kru)*cau)
          Hu(2,i,j) = Hu(1,i,j)              ! aggiorno termine noto !
        end do

        do ij = 1,noradi                            ! coco nord u !
          j = my
          hhh = proplo(i,j)
          greta = zer                                ! eta0 cost !
          hhh = hhh**(set/tre)
          cau = (dx(i-1)*cappa(i,j-1)+dx(i)*cappa(i-1,j-1))/
     $         (dx(i)+dx(i-1))
          u(i,j) = hhh*( u(i,j) + dt*(gam(kru)*Hu(1,i,j)+
     $                          roh(kru)*Hu(2,i,j)
```

```
     $                          -alf(kru)*greta) )/
     $                        (hhh+dt*alf(kru)*cau)
         Hu(2,i,j) = Hu(1,i,j)              ! aggiorno termine noto !
       end do
      end do

..... aggiorno v .....*

      do i = 1,mx1
       do j = 2,my1
        hhh = (dy(j-1)*prof(i,j)+dy(j)*prof(i,j-1))/(dy(j-1)+dy(j))
        greta = gra*hhh*(seta(i,j)-seta(i,j-1))*due/(dy(j-1)+dy(j))
        hhh = hhh*(  (dy(j-1)*prof(i,j)+dy(j)*prof(i,j-1))/
     $             (dy(j-1)+dy(j))  )**(qua/tre)
        hhh = hhh**(set/tre)
        cav = (dy(j-1)*cappa(i,j)+dy(j)*cappa(i,j-1))/(dy(j)+dy(j-1))
        Hv(1,i,j) = Hv(1,i,j)
     $            - saspi(i,j)*(v(i,j)-v(i,my))        ! spi num !
        v(i,j) = float(iw_v(i,j))*
     $          hhh*( v(i,j) + dt*(gam(kru)*Hv(1,i,j)+
     $                          roh(kru)*Hv(2,i,j)
     $                         -alf(kru)*greta) )/
     $                         (hhh+dt*alf(kru)*cav)
        Hv(2,i,j) = Hv(1,i,j)              ! aggiorno termine noto !
       end do

       j = my                                  ! coco nord v !
       do ij = 1,noradi
        hhh = umz*(proplo(i,j)+proplo(i+1,j))
        greta = gra*hhh*(ott*seta0-ove*seta(i,j-1)+seta(i,j-2))/
     $                (tre*dy(j-1))            ! dy(my1)=dy(my2) !
        hhh = hhh**(set/tre)
        cav = cappa(i,j-1)
        v(i,j) = hhh*( v(i,j) + dt*(gam(kru)*Hv(1,i,j)+
     $                          roh(kru)*Hv(2,i,j)
     $                         -alf(kru)* hhh) )/
     $                         (hhh+dt*alf(kru)*cav)
        Hv(2,i,j) = Hv(1,i,j)              ! aggiorno termine noto !
       end do
       do ij = 1,(1-noradi)
        v(i,j) = Q1/abs(x(1,1)-x(mx,1))
       end do
      end do

.....free slip varie.....
      do k = 1,(1-lip_w)                       ! if(lip_w.eq.0) !
       do j = 2,my
```

```
          v(0,j) = v(1,j)
          vplo(1,j) = v(0,j)/proplo(1,j)
        end do
      end do
      do k = 1,(1-lip_e)                    ! if(lip_e.eq.0) !
        do j = 2,my
          v(mx,j) = v(mx1,j)
          vplo(mx,j) = v(mx,j)/proplo(mx,j)
        end do
      end do
      do k = 1,(1-lip_s)                    ! if(lip_s.eq.0) !
        do i = 2,mx1
          u(i,0) = u(i,1)
          uplo(i,1) = u(i,0)/proplo(i,1)
        end do
      end do

.....*
 227  continue                             ! fine substeps Ru-Ku !
.....*

      call etano(seta,setaplo,seta0,mx,my,dx,dy,noradi) ! eta nei nodi !

      call vorti(vor,u,v,proplo,dx,dy,zer,uqt,umz,tre,ott,ove,mx,my,
     $           mx1,my1,mx2,my2,lip_w,lip_e,lip_s)

..... Monitoraggi vari SASHA .....
      recel = zer
      cfl = zer
      dm_0 = zer
      dm_1 = zer
      dm_2 = zer
      scamu = zer
      scamv = zer
      do j = 1,my1
        do i = 1,mx1
          rex = u(i,j)*dx(i)/
     $           (viva(i,j)*umz*(proplo(i,j)+proplo(i,j+1)))
          rey = v(i,j)*dy(j)/
     $           (viva(i,j)*umz*(proplo(i,j)+proplo(i+1,j)))
          recel = max(recel,rex,rey)
          hhh = sqrt(gra*prof(i,j))
          cf = ((abs(u(i,j))*due/(proplo(i,j)+proplo(i,j+1))+hhh)/dx(i)
     $         +(abs(v(i,j))*due/(proplo(i,j)+proplo(i+1,j))+hhh)/dy(j)
     $         )*dt
          cfl = max(cf,cfl)
          div = abs((u(i+1,j)-u(i,j))/dx(i)+(v(i,j+1)-v(i,j))/dy(j))
```

```
        dm_0 = max(div,dm_0)
        div = abs((u(i+1,j)-u(i,j))/dx(i)+(v(i,j+1)-v(i,j))/dy(j)
     $            +(seta(i,j)-setav(i,j))/dt)
        dm_1 = max(div,dm_1)
        div = abs((u(i+1,j)-u(i,j))/dx(i)+(v(i,j+1)-v(i,j))/dy(j)
     $       +(tre*seta(i,j)-qua*setav(i,j)+setavv(i,j))/(due*dt))
        dm_2 = max(div,dm_2)
        scau = abs(u(i,j)-uv(i,j))/
     $            (umz*(proplo(i,j)+proplo(i,j+1))*dt)
        scamu = max(scau,scamu)
        scav = abs(v(i,j)-vv(i,j))/
     $            (umz*(proplo(i,j)+proplo(i+1,j))*dt)
        scamv = max(scav,scamv)
        uv(i,j) = u(i,j)
        vv(i,j) = v(i,j)
        setavv(i,j) = setav(i,j)
        setav(i,j) = seta(i,j)
      end do
     end do
     scamm = max(scamu,scamv)

-----------------------------------------------------------------------
     do j = 2,my1
       do i = 2,mx1
        uplo(i,j) = float(iw_plo(i,j))*
     $            (dy(j-1)*u(i,j)+dy(j)*u(i,j-1))/
     $                  ((dy(j)+dy(j-1))*proplo(i,j))
        vplo(i,j) = float(iw_plo(i,j))*
     $            (dx(i-1)*v(i,j)+dx(i)*v(i-1,j))/
     $                  ((dx(i)+dx(i-1))*proplo(i,j))
       end do
     end do
     j = my                                  ! coco nord !
     do i = 2,mx1
       uplo(i,j) = float(iw_plo(i,j))*u(i,j)/proplo(i,j)
       vplo(i,j) = float(iw_plo(i,j))*
     $   (dx(i-1)*v(i,j)+dx(i)*v(i-1,j))/((dx(i)+dx(i-1))*proplo(i,j))
     end do
-----------------------------------------------------------------------
     Q2 = zer
     do i = istre_w+1,mx1-istre_e
       Q2 = Q2 + v(i,my-jstre)*dx(i)
     end do
     Q3 = zer
     do j = 1,116!my1-jstre                  !!!!!!!!!!!!!!
       Q3 = Q3 + u(istre_w+31,j)*dy(j)       !!!!!!!!!!!!!!
     end do
```

```
      Q4 = zer
      do j = 162,193!1,my1-jstre                    !!!!!!!!!!!!!!
        Q4 = Q4 + u(istre_w+148,j)*dy(j)            !!!!!!!!!!!!!!
        Q4 = Q4 + u(mx-istre_e,j)*dy(j)
      end do
      Bilancio = (Q1-Q2+Q3-Q4)/Q1
-----------------------------------------------------------------------
     if(mod(iter,kpri).eq.0) then
       write(44,3333) sngl(time),sngl(dm_0),sngl(dm_1),sngl(dm_2),
     $   sngl(scamu),sngl(scamv),
     $   sngl(viscolami*(uno-float(iter)/(float(maxite)+.0001d0)))
 3333   format(1x,7(g10.4,1x))
       write(*,1974) iter,dm_2,cfl,scamm,recel
       write(19,1975) iter,dm_2,cfl,scamu,scamv,recel
 1974   format(/,1x,i6,' div:',g8.2,'  cfl:',f9.4,
     $   '   sc. max:',g8.2,' rec:',g8.2)
 1975   format(1x,i6,' div:',g8.2,' cfl:',f9.4,
     $   ' sc. u:',g8.2,' sc. v:',g8.2,'rec:',g8.2)
      end if

.....................................................................*
..... PASSAGGI SASHA/SLUW .....................................*
      do ijk = 1,(1-injet)
.....................................................................*
      do i = 1,nplib
        j = 1 + (i-1)/ntra
        k = i - (j-1)*ntra
        usa(i) = smorz(i)* uplo *(u(j,k)+u(j+1,k))/prof(j,k)
        vsa(i) = smorz(i)*umz*(v(j,k)+v(j,k+1))/prof(j,k)
        disa(i) = (u(j+1,k)-u(j,k))/dx(j) + (v(j,k+1)-v(j,k))/dy(k)
     $         + ((u(j+1,k)-u(j,k))/dx(j) + (v(j,k+1)-v(j,k))/dy(k))*
     $           seta(j,k)/prof(j,k)
        disa(i) = smorz(i)*disa(i)
        etasa(i) = smorz(i)*seta(j,k)
        vivasa(i) = viva(j,k)
      end do
      do i = 1,npan
        k = lat + 1 + (i-1)/lat                  ! careful: lat=long !
        j = lat/2 + i - (k-lat-1)*lat
        usama(i) = umz*(u(j,k)+u(j+1,k))/prof(j,k)
        vsama(i) = umz*(v(j,k)+v(j,k+1))/prof(j,k)
      end do
**************************************************************************
      end do                              ! if(isasha.eq.1) sasha va !
**************************************************************************
.....................................................................*
.....................................................................*
```

```
.....*
 Ciclo Runge--Kutta SLUW
.....*
     do ik = 1,ikutta
       trk = time + dt*rk(ik,2)
      ---------------------------------------------------------
      Configurazione Intermedia Runge--Kutta
      ---------------------------------------------------------
       call rukut1 (nplib,etan,fin,eta,teta,fi,tofi,rk,ik,dt)
      ---------------------------------------------------------
      Soluzione Equazione Integrale
      ---------------------------------------------------------
 Termine noto
       do i=1,npan
         call Wave2 (Am,Wn,Om,gra,hmax,trk,ym(i),zm(i),phase,
    $              fiw,uw,vw,ww)
         sigma(i) = -((uw+usama(i))*cosxn(i)
    $              +(vw+vsama(i))*cosyn(i)
    $                        +ww*coszn(i))
         sigma(i) = float((1-iatto(i)))*sigma(i)
       end do
       do i=npan+1,npan+nplib
         sigma(i) = fin(i-npan)
       end do

       call dgesl(a,nnnn,nnnn,indx,sigma,0)

 Velocita Superficie libera
       do 503 i=npan+1,npan+nplib          ! fiz superficie libera !
        somx=zer
        somy=zer
        somz=zer
        do k=1,npan
          somx=somx+sigma(k)*vx(i-npan,k)
          somy=somy+sigma(k)*vy(i-npan,k)
          somz=somz+sigma(k)*vz(i-npan,k)
        end do
        do 523 k=1,nplib
          somx=somx+sigma(k+npan)*tx(i-npan,k)
          somy=somy+sigma(k+npan)*ty(i-npan,k)
          somz=somz+sigma(k+npan)*tz(i-npan,k)
523       continue
        fix(i)=somx
        fiy(i)=somy
        fiz(i)=somz
503    continue
-------------------------------------------------------------
```

```
      Campo ondoso incidente e derivate elevazione d'onda
-------------------------------------------------------------
        call Wave (npan,ym,zm,phase,nplib,
     $             yn,eta0,eta0y,fi0y,fi0z,Am,Wn,Om,hmax,gra,trk)
        call derita(ntra,nlo,nplib,eta,etax,etay,etasa,etasax,etasay,
     $              r2dx,r2dy,usa,vsa)
-------------------------------------------------------------
         Accumula correzione Runge-Kutta
-------------------------------------------------------------
        call rukut2(npan,nplib,deta,dfi,etan,fin,fix,fiy,fiz,
     $              etax,etay,fi0y,eta0y,usa,vsa,disa,
     $              etasax,etasay,spinu,tofi,teta,gra,rk,ik)
      end do
.....................................................................*
           Fine Ciclo Runge--Kutta SLUW
.....................................................................*

      call rukut3 (nplib,eta,fi,deta,dfi,dt) ! Nuova conf. passo succ. !

-----------------------------------------------------------------------
           TAGLI TRASVERSALI E LONGITUDINALI
-----------------------------------------------------------------------
      write(cha,99) iter

      nfile= label//'vall1'//cha//'.dat'              ! secondo taglio !
      open(unit=71,file=nfile,status='unknown')
      rewind(71)
      do i = i1,nplib-ntra+i1,ntra
        etata = eta(i-1)+eta0(i-1)+etasa(i-1) +
     $         (eta(i)+eta0(i)+etasa(i)-eta(i-1)-eta0(i-1)-etasa(i-1))
     $        /(yn(i)-yn(i-1))*(yta1-yn(i-1))
        write(71,*) xn(i),etata
      end do
      close(71)

      nfile= label//'vall2'//cha//'.dat'              ! terzo taglio !
      open(unit=71,file=nfile,status='unknown')
      rewind(71)
      do i = i2,nplib-ntra+i2,ntra
        etata = eta(i-1)+eta0(i-1)+etasa(i-1) +
     $         (eta(i)+eta0(i)+etasa(i)-eta(i-1)-eta0(i-1)-etasa(i-1))
     $        /(yn(i)-yn(i-1))*(yta2-yn(i-1))
        write(71,*) xn(i),etata
      end do
      close(71)

      nfile= label//'yeta'//cha
```

```
      open(unit=20,file=nfile,status='unknown')              ! talon !
      rewind(20)
      do k = nplib/2+ntra,nplib/2+1,-1
        write(20,*) sngl(-yn(k)),sngl(eta(k)),
        write(20,789) sngl(yn(k)),sngl(eta(k)),
     $                sngl(eta0(k)),
     $                sngl(eta0(k)+eta(k)),
     $                sngl(eta0(k)+eta(k)+etasa(k))
      end do
 789  format(5(1x,f12.6))
      close(20)

.....................................................................
      end do                        ! injet !
.....................................................................
-----------------------------------------------------------------------
         PLOTTAGGI VARI
-----------------------------------------------------------------------
      if(mod(iter,ksta).eq.0) then
.....................................................................*
      do ijk = 1,injet
        open(unit=94,file='contwr.dat',form='unformatted')
        rewind(94)
         write(94) ((uplo(i,j),i=1,mx),j=1,my),
     $            ((vplo(i,j),i=1,mx),j=1,my),
     $            ((setaplo(i,j),i=1,mx),j=1,my),
     $            ((u(i,j),i=1,mx),j=0,my),
     $            ((v(i,j),i=0,mx),j=1,my),
     $            ((seta(i,j),i=1,mx1),j=1,my1),
     $            ((viva(i,j),i=1,mx1),j=1,my1)
        close(94)
      end do
.....................................................................*
        ist = iter/ksta
        write(cha,99) ist
      do ijk = 1,(1-injet)
        nfile= label//'gris'//cha//'.dat'
        open (99,file=nfile,form='unformatted')  ! griglia plot3d SLUW !
        rewind(99)
        write(99) ntra,nlo,1
        write(99) (sngl(xn(i)),i=1,nplib),
     $            (sngl(yn(i)),i=1,nplib),
     $            (sngl(eta0(i)+eta(i)+etasa(i)),i=1,nplib)
        close(99)
        nfile= label//'grit'//cha//'.dat'
        open (99,file=nfile,form='unformatted')  ! griglia plot3d SLUW !
        rewind(99)
```

```
      write(99) ntra,nlo,1
      write(99) (sngl(xn(i)),i=1,nplib),
     $          (sngl(yn(i)),i=1,nplib),
     $          (sngl(eta0(i)+eta(i)),i=1,nplib)
      close(99)
      nfile= label//'grip'//cha//'.dat'
      open (99,file=nfile,form='unformatted')  ! griglia plot3d SLUW !
      rewind(99)
      write(99) ntra,nlo,1
      write(99) (sngl(xn(i)),i=1,nplib),
     $          (sngl(yn(i)),i=1,nplib),
     $          (sngl(eta(i)),i=1,nplib)
      close(99)

       nfile= label//'liv'//cha//'.dat'
       open(99,file=nfile,form='unformatted')
            rewind(99)
            write(99) ntra,nlo,1
            write(99) 0.,0.,0.,0.
            write (99)(1.        ,i=1,nplib),
     $                 (sngl(eta(i))   ,i=1,nplib),
     $                 (sngl(eta(i)+etasa(i))   ,i=1,nplib),
     $                 (sngl(eta0(i)+eta(i)+etasa(i))   ,i=1,nplib),
     $                 (sngl(spinu(i)) ,i=1,nplib)
       close(99)
       nfile= label//'uvw'//cha//'.dat'
       open(99,file=nfile,form='unformatted')
            rewind(99)
            write(99) ntra,nlo,1
            write(99) 0.,0.,0.,0.
            write (99)(1.        ,i=1,nplib),
     $              (sngl(usa(i)+fix(npan+i)),i=1,nplib),
     $              (sngl(vsa(i)+fi0y(npan+i)+fiy(npan+i)),i=1,nplib),
     $              (sngl(fi0z(npan+i)+fiz(npan+i)),i=1,nplib),
     $              (sngl(vivasa(i)),i=1,nplib)
       close(99)
     end do                              ! injet !

     do ijk = 1,injet
      nfile=label//'insa'//cha//'.dat'
     end do
     do ijk = 1,(1-injet)
      nfile=label//'sa'//cha//'.dat'
     end do
      open(59,file=nfile,form='unformatted')
      rewind(59)
      write(59) mx,my
```

```
      write(59) 1., 1., sngl(Reynolds), sngl(time)
      write(59)
    %     ((1.,i=1,mx),j=1,my),
    %     ((sngl(uplo(i,j)),i=1,mx),j=1,my),
    %     ((sngl(vplo(i,j)),i=1,mx),j=1,my),
    %     ((sngl(setaplo(i,j)),i=1,mx),j=1,my)
      close(59)

      nfile=label//'ax'//cha//'.dat'
      open(59,file=nfile,status='unknown')
      rewind(59)
        do i = 1,mx
          write(59,95) sngl(x(i,1)),
     $                 sngl(uplo(i,my1/2+1)),
     $                  sngl(vplo(i,my1/2+1)),
     $                  sngl(setaplo(i,my1/2+1))
        end do
95        format(4(1x,f14.8))
      close(59)
      nfile=label//'ay'//cha//'.dat'
      open(59,file=nfile,status='unknown')
      rewind(59)
        i = (ibos+ibod)/2
        do j = 1,my
          write(59,95) sngl(y(1,j)),
     $                 sngl(uplo(i,j)),
     $                 sngl(vplo(i,j)),
     $                 sngl(setaplo(i,j)),
     $                 sngl(saspi(i,j))
      end do
95      format(5(1x,f14.8))
      close(59)
    end if
c-----------------------------------------------------------------------


 777  continue    ! FINE CICLO TEMPORALE !


    close(19)
    close(44)
99   format(i4.4)


c_____________________________________________________________________
    stop 'VISPO: Fine Elaborazione'
    end
```

```
      subroutine martufo(nplib,lat,long,npan,xn,yn,zn,
     $        x1,x2,x3,x4,xh,y1,y2,y3,y4,yh,z1,z2,z3,z4,zh,diana,
     $        area,cosxn,cosyn,coszn,amat,hmax,iatto)
```

```
      integer*4 npan,nplib,lat,long
      real*8 xn(*),yn(*),zn(*),
     $     x1(*),x2(*),x3(*),x4(*),xh(*),
     $     y1(*),y2(*),y3(*),y4(*),yh(*),
     $     z1(*),z2(*),z3(*),z4(*),zh(*),
     $     cosxn(*),cosyn(*),coszn(*),area(*),
     $     amat(npan+nplib,*),
     $     atr(3,3),diana,hmax

      real*8 pig,eps,hemips,upiep
      integer*4 lat_f,long_f,i,k,iikk,iatto(*)
      real*8    tre,det_a,det_b,det_c,absa,absb,absc,dem,a,b,c,d
      real*8    sud,sden,deta,detb,detc
      real*8    xx1,xx2,xx3,xx4
      real*8    yy1,yy2,yy3,yy4
      real*8    zz1,zz2,zz3,zz4
      real*8    a1,b1,c1,d1
      real*8    a2,b2,c2,d2
      real*8    a3,b3,c3,d3
      real*8    a4,b4,c4,d4
      real*8    d123,d134,d124
      real*8    abs123,abs134,abs124
      real*8    aax,aay,aaz,aaa
      real*8    bbx,bby,bbz
      real*8    drelpq,dilato
      real*8    x12,x13,x23,x34,x41
      real*8    y12,y13,y23,y34,y41
      real*8    z12,z13,z23,z34,z41
      real*8    w123,w341
      real*8    fi,u,v,w,uu,vv,ww

      pig=4.0*atan(1.0)
      eps = 0.00010
1234  eps = eps*0.5
      upiep = 1.0 + eps
      if(upiep.ne.1.0) go to 1234
      eps = eps*2.0
      hemips = eps
```

```
129  hemips = hemips*0.5
     if((hemips+1.0).ne.1.0) go to 129
     hemips = hemips*2.0

c-----------------------------------------------------
c    Lettura dei punti di confine
c-----------------------------------------------------
     open(unit=22,file='martufo.dat',status='unknown')
       rewind(22)
       read (22,*) lat_f , long_f
       if(lat_f.ne.lat.or.long_f.ne.long) then
         write(*,*)'Disaccordo tra il file martufo.dat e sluw.f'
         write(*,*)'Lat o long si riferiscono ad un caso diverso'
         stop
       end if
       do i=1,npan
         read (22,*) x1(i),x2(i),x3(i),x4(i),xn(i)
       end do
       do i=1,npan
         read (22,*) y1(i),y2(i),y3(i),y4(i),yn(i)
       end do
       do i=1,npan
         read (22,*) z1(i),z2(i),z3(i),z4(i),zn(i)
       end do
     close(22)
100  format(1x,a)
     open(unit=22,file='stabody.dat',status='unknown')
     rewind(22)
       do i = 1,npan
         write(22,*) xn(i),yn(i),zn(i)
       end do
     close(22)


c--------------------------------------------------------------------
     do i = 1,npan                                          !
c--------------------------------------------------------------------
    Minimi quadrati: piano medio                                    !
c--------------------------------------------------------------------
       atr(1,1) = (x1(i)-xn(i))**2+(x2(i)-xn(i))**2+
     $            (x3(i)-xn(i))**2+(x4(i)-xn(i))**2
       atr(2,2) = (y1(i)-yn(i))**2+(y2(i)-yn(i))**2+
     $            (y3(i)-yn(i))**2+(y4(i)-yn(i))**2
       atr(3,3) = (z1(i)-zn(i))**2+(z2(i)-zn(i))**2+
     $            (z3(i)-zn(i))**2+(z4(i)-zn(i))**2
       atr(1,2) = (x1(i)-xn(i))*(y1(i)-yn(i))+
     $            (x2(i)-xn(i))*(y2(i)-yn(i))+
```

```
$            (x3(i)-xn(i))*(y3(i)-yn(i))+
$            (x4(i)-xn(i))*(y4(i)-yn(i))
   atr(1,3) = (x1(i)-xn(i))*(z1(i)-zn(i))+
$            (x2(i)-xn(i))*(z2(i)-zn(i))+
$            (x3(i)-xn(i))*(z3(i)-zn(i))+
$            (x4(i)-xn(i))*(z4(i)-zn(i))
   atr(2,3) = (y1(i)-yn(i))*(z1(i)-zn(i))+
$            (y2(i)-yn(i))*(z2(i)-zn(i))+
$            (y3(i)-yn(i))*(z3(i)-zn(i))+
$            (y4(i)-yn(i))*(z4(i)-zn(i))
   atr(2,1) = atr(1,2)
   atr(3,1) = atr(1,3)
   atr(3,2) = atr(2,3)
   tre = atr(1,1)*atr(2,2)*atr(3,3) + atr(1,2)*atr(2,3)*atr(3,1) +
$       atr(1,3)*atr(2,1)*atr(3,2) - atr(3,1)*atr(2,2)*atr(1,3) -
$       atr(3,2)*atr(2,3)*atr(1,1) - atr(3,3)*atr(2,1)*atr(1,2)
   if(abs(tre).ge.eps) then
  write(*,*)'*****************************************************'
  write(*,*) ' PANNELLO ',i,'         det 3x3 =',tre
   end if

   det_a = atr(2,2)*atr(3,3) - atr(3,2)*atr(2,3)
   det_b = atr(1,1)*atr(3,3) - atr(3,1)*atr(1,3)
   det_c = atr(1,1)*atr(2,2) - atr(2,1)*atr(1,2)
   absa = abs(det_a)
   absb = abs(det_b)
   absc = abs(det_c)
   dem = max(absa,absb,absc)

   if(dem.eq.absa) then
    sud = 1.0/det_a
    a = 1.0
    b = (-atr(2,1)*atr(3,3) + atr(3,1)*atr(2,3))*sud
    c = (-atr(2,2)*atr(3,1) + atr(3,2)*atr(2,1))*sud
   else if(dem.eq.absb) then
    sud = 1.0/det_b
    a = (-atr(1,2)*atr(3,3) + atr(3,2)*atr(1,3))*sud
    b = 1.0
    c = (-atr(1,1)*atr(3,2) + atr(3,1)*atr(1,2))*sud
   else if(dem.eq.absc) then
    sud = 1.0/det_c
    a = (-atr(1,3)*atr(2,2) + atr(2,3)*atr(1,2))*sud
    b = (-atr(1,1)*atr(2,3) + atr(2,1)*atr(1,3))*sud
    c = 1.0
   end if
   d = -(a*xn(i)+b*yn(i)+c*zn(i))
```

```
c-----------------------------------------------------------------------
      Costruzione del pannello                                !
c-----------------------------------------------------------------------
        deta = a*a*a + a*b*b + a*c*c
        detb = a*a*b + b*b*b + b*c*c
        detc = - (a*a*c + b*b*c + c*c*c)
        absa = abs(deta)
        absb = abs(detb)
        absc = abs(detc)
        dem = max(absa,absb,absc)
        if(absa.eq.dem) then
          sud = 1.0/deta
c         write(*,*)' CASO a',' SUD=',sud
          xx1 = (-a*a*d + a*c*(c*x1(i)-a*z1(i))
     %              + a*b*(b*x1(i)-a*y1(i))) * sud
          yy1 = (-a*a*(b*x1(i)-a*y1(i)) + b*c*(c*x1(i)-a*z1(i))
     $          - c*c*(b*x1(i)-a*y1(i)) - a*b*d) * sud
          zz1 = (-a*a*(c*x1(i)-a*z1(i)) + b*c*(b*x1(i)-a*y1(i))
     %          - b*b*(c*x1(i)-a*z1(i)) - a*c*d) * sud
          xx2 = (-a*a*d + a*c*(c*x2(i)-a*z2(i))
     %              + a*b*(b*x2(i)-a*y2(i))) * sud
          yy2 = (-a*a*(b*x2(i)-a*y2(i)) + b*c*(c*x2(i)-a*z2(i))
     $          - c*c*(b*x2(i)-a*y2(i)) - a*b*d) * sud
          zz2 = (-a*a*(c*x2(i)-a*z2(i)) + b*c*(b*x2(i)-a*y2(i))
     %          - b*b*(c*x2(i)-a*z2(i)) - a*c*d) * sud
          xx3 = (-a*a*d + a*c*(c*x3(i)-a*z3(i))
     %              + a*b*(b*x3(i)-a*y3(i))) * sud
          yy3 = (-a*a*(b*x3(i)-a*y3(i)) + b*c*(c*x3(i)-a*z3(i))
     $          - c*c*(b*x3(i)-a*y3(i)) - a*b*d) * sud
          zz3 = (-a*a*(c*x3(i)-a*z3(i)) + b*c*(b*x3(i)-a*y3(i))
     %          - b*b*(c*x3(i)-a*z3(i)) - a*c*d) * sud
          xx4 = (-a*a*d + a*c*(c*x4(i)-a*z4(i))
     %              + a*b*(b*x4(i)-a*y4(i))) * sud
          yy4 = (-a*a*(b*x4(i)-a*y4(i)) + b*c*(c*x4(i)-a*z4(i))
     $          - c*c*(b*x4(i)-a*y4(i)) - a*b*d) * sud
          zz4 = (-a*a*(c*x4(i)-a*z4(i)) + b*c*(b*x4(i)-a*y4(i))
     %          - b*b*(c*x4(i)-a*z4(i)) - a*c*d) * sud
        else if(absb.eq.dem) then
          sud = 1.0/detb
          xx1 = (-d*a*b + c*(b*x1(i)-a*y1(i))*c +
     %          (c*y1(i)-b*z1(i))*a*c + b*(b*x1(i)-a*y1(i))*b) * sud
          yy1 = (-a*(b*x1(i)-a*y1(i))*b + c*b*(c*y1(i)-b*z1(i))
     %          - b*b*d) * sud
          zz1 = (-a*a*(c*y1(i)-b*z1(i)) - d*b*c -
     %          c*(b*x1(i)-a*y1(i))*a - (c*y1(i)-b*z1(i))*b*b) * sud
          xx2 = (-d*a*b + c*(b*x2(i)-a*y2(i))*c +
     %          (c*y2(i)-b*z2(i))*a*c + b*(b*x2(i)-a*y2(i))*b) * sud
```

```
      yy2 = (-a*(b*x2(i)-a*y2(i))*b + c*b*(c*y2(i)-b*z2(i))
     %         - b*b*d) * sud
      zz2 = (-a*a*(c*y2(i)-b*z2(i)) - d*b*c -
     %         c*(b*x2(i)-a*y2(i))*a - (c*y2(i)-b*z2(i))*b*b) * sud
      xx3 = (-d*a*b + c*(b*x3(i)-a*y3(i))*c +
     %         (c*y3(i)-b*z3(i))*a*c + b*(b*x3(i)-a*y3(i))*b) * sud
      yy3 = (-a*(b*x3(i)-a*y3(i))*b + c*b*(c*y3(i)-b*z3(i))
     %         - b*b*d) * sud
      zz3 = (-a*a*(c*y3(i)-b*z3(i)) - d*b*c -
     %         c*(b*x3(i)-a*y3(i))*a - (c*y3(i)-b*z3(i))*b*b) * sud
      xx4 = (-d*a*b + c*(b*x4(i)-a*y4(i))*c +
     %         (c*y4(i)-b*z4(i))*a*c + b*(b*x4(i)-a*y4(i))*b) * sud
      yy4 = (-a*(b*x4(i)-a*y4(i))*b + c*b*(c*y4(i)-b*z4(i))
     %         - b*b*d) * sud
      zz4 = (-a*a*(c*y4(i)-b*z4(i)) - d*b*c -
     %         c*(b*x4(i)-a*y4(i))*a - (c*y4(i)-b*z4(i))*b*b) * sud
    else if(absc.eq.dem) then
      sud = 1.0/detc
      xx1 = (d*c*a - b*b*(c*x1(i)-a*z1(i)) -
     $         (c*x1(i)-a*z1(i))*c*c + a*(c*y1(i)-b*z1(i))*b) * sud
      yy1 = (- a*(c*y1(i)-b*z1(i))*a + d*b*c -
     %         c*(c*y1(i)-b*z1(i))*c + (c*x1(i)-a*z1(i))*b*a) * sud
      zz1 = (a*c*(c*x1(i)-a*z1(i)) +
     %         b*(c*y1(i)-b*z1(i))*c + c*c*d) * sud
      xx2 = (d*c*a - b*b*(c*x2(i)-a*z2(i)) -
     $         (c*x2(i)-a*z2(i))*c*c + a*(c*y2(i)-b*z2(i))*b) * sud
      yy2 = (- a*(c*y2(i)-b*z2(i))*a + d*b*c -
     %         c*(c*y2(i)-b*z2(i))*c + (c*x2(i)-a*z2(i))*b*a) * sud
      zz2 = (a*c*(c*x2(i)-a*z2(i)) +
     %         b*(c*y2(i)-b*z2(i))*c + c*c*d) * sud
      xx3 = (d*c*a - b*b*(c*x3(i)-a*z3(i)) -
     $         (c*x3(i)-a*z3(i))*c*c + a*(c*y3(i)-b*z3(i))*b) * sud
      yy3 = (- a*(c*y3(i)-b*z3(i))*a + d*b*c -
     %         c*(c*y3(i)-b*z3(i))*c + (c*x3(i)-a*z3(i))*b*a) * sud
      zz3 = (a*c*(c*x3(i)-a*z3(i)) +
     %         b*(c*y3(i)-b*z3(i))*c + c*c*d) * sud
      xx4 = (d*c*a - b*b*(c*x4(i)-a*z4(i)) -
     $         (c*x4(i)-a*z4(i))*c*c + a*(c*y4(i)-b*z4(i))*b) * sud
      yy4 = (- a*(c*y4(i)-b*z4(i))*a + d*b*c -
     %         c*(c*y4(i)-b*z4(i))*c + (c*x4(i)-a*z4(i))*b*a) * sud
      zz4 = (a*c*(c*x4(i)-a*z4(i)) +
     %         b*(c*y4(i)-b*z4(i))*c + c*c*d) * sud
    end if
    x1(i) = xx1
    y1(i) = yy1
    z1(i) = zz1
    x2(i) = xx2
```

```
      y2(i) = yy2
      z2(i) = zz2
      x3(i) = xx3
      y3(i) = yy3
      z3(i) = zz3
      x4(i) = xx4
      y4(i) = yy4
      z4(i) = zz4

!--------------------------------------------------------------------
!  calcolo di xh,yh,zh, area e coseni direttori                      !
!--------------------------------------------------------------------
      a1 = x4(i)-x3(i)
      b1 = y4(i)-y3(i)
      c1 = z4(i)-z3(i)
      d1 = x1(i)*(x4(i)-x3(i)) + y1(i)*(y4(i)-y3(i)) +
     $     z1(i)*(z4(i)-z3(i))
      a2 = y4(i)-y3(i)
      b2 = x3(i)-x4(i)
      c2 = 0.0
      d2 = x3(i)*(y4(i)-y3(i)) + y3(i)*(x3(i)-x4(i))
      a3 = 0.0
      b3 = z4(i)-z3(i)
      c3 = y3(i)-y4(i)
      d3 = y3(i)*(z4(i)-z3(i)) + z3(i)*(y3(i)-y4(i))
      a4 = z4(i)-z3(i)
      b4 = 0.0
      c4 = x3(i)-x4(i)
      d4 = x3(i)*(z4(i)-z3(i)) + z3(i)*(x3(i)-x4(i))
      d123 = a1*b2*c3 + b1*c2*a3 + c1*a2*b3 -a3*b2*c1 -
     $     b3*c2*a1 - c3*a2*b1
      d124 = a1*b2*c4 + b1*c2*a4 + c1*a2*b4 -a4*b2*c1 -
     $     b4*c2*a1 - c4*a2*b1
      d134 = a1*b3*c4 + b1*c3*a4 + c1*a3*b4 -a4*b3*c1 -
     $     b4*c3*a1 - c4*a3*b1
      abs123 = abs(d123)
      abs124 = abs(d124)
      abs134 = abs(d134)
      dem = max(abs123,abs124,abs134)
      if(abs123.eq.dem) then
        sud = 1.0/d123
        xh(i) = (d1*b2*c3 + b1*c2*d3 + c1*d2*b3 -d3*b2*c1 -
     $          b3*c2*d1 - c3*d2*b1) * sud
        yh(i) = (a1*d2*c3 + d1*c2*a3 + c1*a2*d3 -a3*d2*c1 -
     $          d3*c2*a1 - c3*a2*d1) * sud
        zh(i) = (a1*b2*d3 + b1*d2*a3 + d1*a2*b3 -a3*b2*d1 -
     $          b3*d2*a1 - d3*a2*b1) * sud
```

```
      else if(abs124.eq.dem) then
        sud = 1.0/d124
        xh(i) = (d1*b2*c4 + b1*c2*d4 + c1*d2*b4 -d4*b2*c1 -
     $          b4*c2*d1 - c4*d2*b1) * sud
        yh(i) = (a1*d2*c4 + d1*c2*a4 + c1*a2*d4 -a4*d2*c1 -
     $          d4*c2*a1 - c4*a2*d1) * sud
        zh(i) = (a1*b2*d4 + b1*d2*a4 + d1*a2*b4 -a4*b2*d1 -
     $          b4*d2*a1 - d4*a2*b1) * sud
      else if(abs134.eq.dem) then
        sud = 1.0/d134
        xh(i) = (d1*b3*c4 + b1*c3*d4 + c1*d3*b4 -d4*b3*c1 -
     $          b4*c3*d1 - c4*d3*b1) * sud
        yh(i) = (a1*d3*c4 + d1*c3*a4 + c1*a3*d4 -a4*d3*c1 -
     $          d4*c3*a1 - c4*a3*d1) * sud
        zh(i) = (a1*b3*d4 + b1*d3*a4 + d1*a3*b4 -a4*b3*d1 -
     $          b4*d3*a1 - d4*a3*b1) * sud
      end if

      aax = x1(i)-x3(i)                              ! area !
      aay = y1(i)-y3(i)
      aaz = z1(i)-z3(i)
      bbx = x4(i)-x2(i)
      bby = y4(i)-y2(i)
      bbz = z4(i)-z2(i)
      aaa = sqrt( (aay*bbz-aaz*bby)**2 +
     &     (aaz*bbx-aax*bbz)**2 + (aax*bby-aay*bbx)**2)
      area(i) = .5*aaa


      sden = 1.0 / sqrt(                        ! normale esterna !
     $ ((y3(i)-y2(i))*(z3(i)-z1(i))-(y4(i)-y1(i))*(z3(i)-z2(i)))**2 +
     $ ((z4(i)-z2(i))*(x3(i)-x1(i))-(z3(i)-z1(i))*(x4(i)-x2(i)))**2 +
     $ ((x3(i)-x2(i))*(y2(i)-y1(i))-(x4(i)-x1(i))*(y3(i)-y2(i)))**2 )
      cosxn(i) = sden*
     $ ((y4(i)-y2(i))*(z3(i)-z1(i))-(y3(i)-y1(i))*(z4(i)-z2(i)))
      cosyn(i) = sden*
     $ ((z4(i)-z2(i))*(x3(i)-x1(i))-(z3(i)-z1(i))*(x4(i)-x2(i)))
      coszn(i) = sden*
     $ ((x4(i)-x2(i))*(y3(i)-y1(i))-(x3(i)-x1(i))*(y4(i)-y2(i)))

-----------------------------------------------------------------------
      iatto(i) = 0
      if(abs((zn(i)-hmax)/hmax).lt.(.00001)) then
        iatto(i) = 1
      end if
-----------------------------------------------------------------------
    end do                                          !
```

```
------------------------------------------------------------------


------------------------------------------------------------------
   Calcolo degli integrali 'corpo-corpo' vxc,vyc,vzc
------------------------------------------------------------------

     do 30 k=1,npan    !!!!! PANNELLO SORG.Q : calcolo p. integraz.

      call prima(x1(k),x2(k),x3(k),x4(k),y1(k),y2(k),y3(k),y4(k),
     $           z1(k),z2(k),z3(k),z4(k),xh(k),yh(k),zh(k),area(k),
     $           w123,w341,dilato,x12,y12,z12,x23,y23,z23,x13,y13,z13,
     $           x34,y34,z34,x41,y41,z41)


      do 40 i=1,npan   !!!!! PANNELLO P

       drelpq=sqrt((xn(i)-xn(k))**2+(yn(i)-yn(k))**2+(zn(i)-zn(k))**2)
     $        / dilato

======================================================================
       if (drelpq.ge.diana) then
======================================================================


-- x,y,z ---

       call splyn(x12,y12,z12,x23,y23,z23,
     $  x13,y13,z13,x34,y34,z34,
     $  x41,y41,z41,xn(i),yn(i),zn(i),w123,w341,fi,u,v,w)

        uu = - u
        vv = - v
        ww = - w

-- x,y,-z ---

       call splyn(x12,y12,-(z12+2.*hmax),x23,y23,-(z23+2.*hmax),
     $  x13,y13,-(z13+2.*hmax),x34,y34,-(z34+2.*hmax),
     $  x41,y41,-(z41+2.*hmax),xn(i),yn(i),zn(i),w123,w341,fi,u,v,w)

        uu = uu - u
        vv = vv - v
        ww = ww - w

======================================================================
       else
```

```
=====================================================================

       if(i.eq.k)then
         iikk=1
       else
         iikk=0
       end if

-- x,y,z ---


       call hemiqua(xn(k),yn(k),zn(k),xn(i),yn(i),zn(i),
     $   x1(k),x2(k),x3(k),x4(k),xh(k),yh(k),zh(k),
     $   y1(k),y2(k),y3(k),y4(k),z1(k),z2(k),z3(k),z4(k),
     $   cosxn(k),cosyn(k),coszn(k),fi,u,v,w,pig,iikk,hemips)

       uu = - u
       vv = - v
       ww = - w

-- x,y,-z ---

       iikk=0
       call hemiqua(xn(k),yn(k),-(zn(k)+2.*hmax),xn(i),yn(i),zn(i),
     $   x1(k),x2(k),x3(k),x4(k),xh(k),yh(k),-(zh(k)+2.*hmax),
     $   y1(k),y2(k),y3(k),y4(k),-(z1(k)+2.*hmax),-(z2(k)+2.*hmax),
     $   -(z3(k)+2.*hmax),-(z4(k)+2.*hmax),
     $   cosxn(k),cosyn(k),-coszn(k),fi,u,v,w,pig,iikk,hemips)

       uu = uu - u
       vv = vv - v
       ww = ww - w

=====================================================================
      end if
=====================================================================
      amat(i,k) = uu*cosxn(i) + vv*cosyn(i) + ww*coszn(i)

 40   continue
 30  continue
     do i = 1,npan
      if(iatto(i).eq.1) then
       do k = 1,npan
         amat(i,k) = 0.0
       end do
       amat(i,i) = 1.0
      end if
```

```
      end do


_____________________________________________________________________
      return
      end
_____________________________________________________________________
_____________________________________________________________________
      subroutine caldic(xa,xb,xc,xd,xx,xq,cosxq,xp,
     $                  ya,yb,yc,yd,yy,yq,cosyq,yp,
     $                  za,zb,zc,zd,zz,zq,coszq,zp,diana,
     $                  arpa,vv,vvx,vvy,vvz,npan,nplib,
     $                  ini,ifi,kin,kfi,hmax,hemips)
c-----------------------------------------------------------------------*
c     Costruzione delle matrici di influenza
c-----------------------------------------------------------------------*

      integer*4 ini,ifi,kin,kfi,npan,nplib
      real*8  diana,hmax,hemips
      real*8  xa(*),xb(*),xc(*),xd(*),xx(*),xq(*),cosxq(*),xp(*),
     $        ya(*),yb(*),yc(*),yd(*),yy(*),yq(*),cosyq(*),yp(*),
     $        za(*),zb(*),zc(*),zd(*),zz(*),zq(*),coszq(*),zp(*),
     $        arpa(*),
     $        vvx(nplib,*),vvy(nplib,*),vvz(nplib,*),vv(npan+nplib,*)

      integer*4 k,i,iikk
      real*8 pig,dilato,drelpq
      real*8 xnt,ynt,znt
      real*8 x12,y12,z12,x23,y23,z23,x13,y13,z13,x34,y34,z34,
     $       x41,y41,z41,w123,w341,fi,u,v,w

c=====================================================================

      pig  = 4.0*atan(1.0)
      iikk = 0

c=====================================================================

      do 100 k=kin,kfi     !!!!! PANNELLO SORG.Q : calcolo p. integraz.

       call prima(xa(k),xb(k),xc(k),xd(k),ya(k),yb(k),yc(k),yd(k),
     $            za(k),zb(k),zc(k),zd(k),xx(k),yy(k),zz(k),arpa(k),
     $            w123,w341,dilato,x12,y12,z12,x23,y23,z23,x13,y13,z13,
     $            x34,y34,z34,x41,y41,z41)

       do 200 i=ini,ifi   !!!!! PANNELLO P

         xnt=xp(i-npan)
```

```
        ynt=yp(i-npan)
        znt=zp(i-npan)
        drelpq=sqrt((xnt-xq(k))**2+(ynt-yq(k))**2+(znt-zq(k))**2)
     $          / dilato

=====================================================================
       if (drelpq.ge.diana) then
=====================================================================


-- x,y,z ---
       call splyn(x12,y12,z12,x23,y23,z23,x13,y13,z13,x34,y34,z34,
     $            x41,y41,z41,xnt,ynt,znt,w123,w341,fi,u,v,w)
        vvx(i-npan,k) = - u
        vvy(i-npan,k) = - v
        vvz(i-npan,k) = - w
        vv(i,k)  = + fi

-- x, y,-z ---
       call splyn(x12,y12,-(z12+2.*hmax),x23,y23,-(z23+2.*hmax),
     $            x13,y13,-(z13+2.*hmax),x34,y34,-(z34+2.*hmax),
     $            x41,y41,-(z41+2.*hmax),xnt,ynt,znt,w123,w341,
     &             fi,u,v,w)
        vvx(i-npan,k) = vvx(i-npan,k) - u
        vvy(i-npan,k) = vvy(i-npan,k) - v
        vvz(i-npan,k) = vvz(i-npan,k) - w
        vv(i,k)  =  vv(i,k) + fi

=====================================================================
       else
=====================================================================
-- x,y,z ---
        call hemiqua
     $       (xq(k),yq(k),zq(k),xnt,ynt,znt,
     $       xa(k),xb(k),xc(k),xd(k),xx(k),yy(k),zz(k),
     $       ya(k),yb(k),yc(k),yd(k),za(k),zb(k),zc(k),zd(k),
     $       cosxq(k),cosyq(k),coszq(k),fi,u,v,w,pig,iikk,hemips)
       vvx(i-npan,k) = - u
       vvy(i-npan,k) = - v
       vvz(i-npan,k) = - w
       vv(i,k)  = + fi

-- x,y,-z ---
        call hemiqua
     $       (xq(k),yq(k),-(zq(k)+2.*hmax),xnt,ynt,znt,
     $        xa(k),xb(k),xc(k),xd(k),xx(k),yy(k),-(zz(k)+2.*hmax),
     $        ya(k),yb(k),yc(k),yd(k),
```

```
     $       -(za(k)+2.*hmax),-(zb(k)+2.*hmax),
     $       -(zc(k)+2.*hmax),-(zd(k)+2.*hmax),
     $        cosxq(k),cosyq(k),-coszq(k),fi,u,v,w,pig,iikk,hemips)
       call splyn(x12,y12,-(z12+2.*hmax),x23,y23,-(z23+2.*hmax),
     $            x13,y13,-(z13+2.*hmax),x34,y34,-(z34+2.*hmax),
     $            x41,y41,-(z41+2.*hmax),xnt,ynt,znt,w123,w341,
     $            fi,u,v,w)
        vvx(i-npan,k) = vvx(i-npan,k) - u
        vvy(i-npan,k) = vvy(i-npan,k) - v
        vvz(i-npan,k) = vvz(i-npan,k) - w
        vv(i,k)  =  vv(i,k) + fi

=====================================================================
       end if
=====================================================================

  200     continue
  100  continue



_____________________________________________________________________
       return
       end
_____________________________________________________________________
_____________________________________________________________________
       subroutine prima(x1,x2,x3,x4,y1,y2,y3,y4,z1,z2,z3,z4,xh,yh,zh,
     $                  area,w123,w341,dilato,
     $                  x12,y12,z12,x23,y23,z23,x13,y13,z13,
     $                  x34,y34,z34,x41,y41,z41)
=====================================================================
    PRIMA
=====================================================================
       real*8 x1,x2,x3,x4,y1,y2,y3,y4,z1,z2,z3,z4,xh,yh,zh,
     $        area,w123,w341,dilato,
     $        x12,y12,z12,x23,y23,z23,x13,y13,z13,
     $        x34,y34,z34,x41,y41,z41
       real*8 dist12,dist23,dist34,dist41,area341,area123,altezza

       dist12  = sqrt ( (x1-x2)**2+(y1-y2)**2+(z1-z2)**2 )
       dist23  = sqrt ( (x2-x3)**2+(y2-y3)**2+(z2-z3)**2 )
       dist34  = sqrt ( (x3-x4)**2+(y3-y4)**2+(z3-z4)**2 )
       dist41  = sqrt ( (x4-x1)**2+(y4-y1)**2+(z4-z1)**2 )
       dilato  = max (dist12,dist23,dist34,dist41)
       altezza = sqrt ( (x1-xh)**2+(y1-yh)**2+(z1-zh)**2 )

       area341 = dist34 * altezza * 0.5
       area123 = area - area341
```

```
      w123    = area123 / 3.0
      w341    = area341 / 3.0

      x12 = 0.5 * ( x1 + x2 )
      y12 = 0.5 * ( y1 + y2 )
      z12 = 0.5 * ( z1 + z2 )
      x23 = 0.5 * ( x2 + x3 )
      y23 = 0.5 * ( y2 + y3 )
      z23 = 0.5 * ( z2 + z3 )
      x13 = 0.5 * ( x1 + x3 )
      y13 = 0.5 * ( y1 + y3 )
      z13 = 0.5 * ( z1 + z3 )
      x34 = 0.5 * ( x3 + x4 )
      y34 = 0.5 * ( y3 + y4 )
      z34 = 0.5 * ( z3 + z4 )
      x41 = 0.5 * ( x4 + x1 )
      y41 = 0.5 * ( y4 + y1 )
      z41 = 0.5 * ( z4 + z1 )



------------------------------------------------------------------
      return
      end
------------------------------------------------------------------
------------------------------------------------------------------
      subroutine hemiqua
     $ (xn,yn,zn,xnt,ynt,znt,xv1,xv2,xv3,xv4,
     $ xh,yh,zh,yv1,yv2,yv3,yv4,zv1,zv2,zv3,zv4,coszitax,
     $ coszitay,coszitaz,vv,vvx,vvy,vvz,pig,iikk,hemips)
------------------------------------------------------------------
------------------------------------------------------------------

      integer*4 i,ii,iikk
      real*8 xn,yn,zn,xnt,ynt,znt,xv1,xv2,xv3,xv4,
     $ xh,yh,zh,yv1,yv2,yv3,yv4,zv1,zv2,zv3,zv4,coszitax,
     $ coszitay,coszitaz,vv,vvx,vvy,vvz,pig,hemips,
     $ x1,x2,x3,x4,y1,y2,y3,y4,xx,yy,zz,
     $ deteta,cosetax,cosetay,cosetaz,
     $ coscsix,coscsiy,coscsiz,vxloc,vyloc,vzloc,radx,rady

      real*8 hsc(4),hss(4),hsdist(4),spicuno(4),spicdue(4),
     %       hsq(4),hsj(4),rpic(4),rr(4),arg1(4),arg2(4)


      radx=sqrt((xv3-xv4)**2+(yv3-yv4)**2+(zv3-zv4)**2)
      coscsix=(-xv4+xv3)/radx
      coscsiy=(-yv4+yv3)/radx
```

```
coscsiz=(-zv4+zv3)/radx
rady=sqrt((xv1-xh)**2+(yv1-yh)**2+(zv1-zh)**2)
cosetax=(xv1-xh)/rady
cosetay=(yv1-yh)/rady
cosetaz=(zv1-zh)/rady

xx=(xnt-xn)*coscsix+(ynt-yn)*coscsiy+(znt-zn)*coscsiz
yy=(xnt-xn)*cosetax+(ynt-yn)*cosetay+(znt-zn)*cosetaz
zz=(xnt-xn)*coszitax+(ynt-yn)*coszitay+(znt-zn)*coszitaz

x1=(xv1-xn)*coscsix+(yv1-yn)*coscsiy+(zv1-zn)*coscsiz
x2=(xv2-xn)*coscsix+(yv2-yn)*coscsiy+(zv2-zn)*coscsiz
x3=(xv3-xn)*coscsix+(yv3-yn)*coscsiy+(zv3-zn)*coscsiz
x4=(xv4-xn)*coscsix+(yv4-yn)*coscsiy+(zv4-zn)*coscsiz
y1=(xv1-xn)*cosetax+(yv1-yn)*cosetay+(zv1-zn)*cosetaz
y2=(xv2-xn)*cosetax+(yv2-yn)*cosetay+(zv2-zn)*cosetaz
y3=(xv3-xn)*cosetax+(yv3-yn)*cosetay+(zv3-zn)*cosetaz
y4=(xv4-xn)*cosetax+(yv4-yn)*cosetay+(zv4-zn)*cosetaz

hsdist(1)=sqrt((x1-x2)**2+(y1-y2)**2)
hsdist(2)=sqrt((x2-x3)**2+(y2-y3)**2)
hsdist(3)=sqrt((x3-x4)**2+(y3-y4)**2)
hsdist(4)=sqrt((x1-x4)**2+(y1-y4)**2)
hsc(1)=(x2-x1)
hsc(2)=(x3-x2)
hsc(3)=(x4-x3)
hsc(4)=(x1-x4)
hss(1)=(y2-y1)
hss(2)=(y3-y2)
hss(3)=(y4-y3)
hss(4)=(y1-y4)
do i = 1,4
  if(hsdist(i).gt.hemips) then
    hsc(i)=hsc(i)/hsdist(i)
    hss(i)=hss(i)/hsdist(i)
  else
    hsc(i)=0.0
    hss(i)=0.0
  end if
end do

spicuno(1)=(x1-xx)*hsc(1)+(y1-yy)*hss(1)
spicdue(1)=(x2-xx)*hsc(1)+(y2-yy)*hss(1)
spicuno(2)=(x2-xx)*hsc(2)+(y2-yy)*hss(2)
spicdue(2)=(x3-xx)*hsc(2)+(y3-yy)*hss(2)
spicuno(3)=(x3-xx)*hsc(3)+(y3-yy)*hss(3)
spicdue(3)=(x4-xx)*hsc(3)+(y4-yy)*hss(3)
```

```
      spicuno(4)=(x4-xx)*hsc(4)+(y4-yy)*hss(4)
      spicdue(4)=(x1-xx)*hsc(4)+(y1-yy)*hss(4)

      rr(1)=(xx-x1)*hss(1)-(yy-y1)*hsc(1)
      rr(2)=(xx-x2)*hss(2)-(yy-y2)*hsc(2)
      rr(3)=(xx-x3)*hss(3)-(yy-y3)*hsc(3)
      rr(4)=(xx-x4)*hss(4)-(yy-y4)*hsc(4)
-----------------------------------------------
      deteta=2.0*pig
      do ii=1,4
        if(rr(ii).lt.0.0) then
          deteta=0.0
          go to 9191
        end if
      end do
 9191 continue
-----------------------------------------------
      rpic(1)=sqrt((xx-x1)**2+(yy-y1)**2+zz**2)
      rpic(2)=sqrt((xx-x2)**2+(yy-y2)**2+zz**2)
      rpic(3)=sqrt((xx-x3)**2+(yy-y3)**2+zz**2)
      rpic(4)=sqrt((xx-x4)**2+(yy-y4)**2+zz**2)

      hsq(1)=log((rpic(1)+rpic(2)+hsdist(1))/(rpic(1)+
     $           rpic(2)-hsdist(1)))
      hsq(2)=log((rpic(2)+rpic(3)+hsdist(2))/(rpic(2)+
     $           rpic(3)-hsdist(2)))
      hsq(3)=log((rpic(3)+rpic(4)+hsdist(3))/(rpic(3)+
     $           rpic(4)-hsdist(3)))
      hsq(4)=log((rpic(4)+rpic(1)+hsdist(4))/(rpic(4)+
     $           rpic(1)-hsdist(4)))

      arg1(1)=rr(1)*abs(zz)*(rpic(1)*spicdue(1)-rpic(2)*spicuno(1))
      arg2(1)=rpic(1)*rpic(2)*(rr(1)**2)+(zz**2)*spicuno(1)*spicdue(1)
      arg1(2)=rr(2)*abs(zz)*(rpic(2)*spicdue(2)-rpic(3)*spicuno(2))
      arg2(2)=rpic(2)*rpic(3)*(rr(2)**2)+(zz**2)*spicuno(2)*spicdue(2)
      arg1(3)=rr(3)*abs(zz)*(rpic(3)*spicdue(3)-rpic(4)*spicuno(3))
      arg2(3)=rpic(3)*rpic(4)*(rr(3)**2)+(zz**2)*spicuno(3)*spicdue(3)
      arg1(4)=rr(4)*abs(zz)*(rpic(4)*spicdue(4)-rpic(1)*spicuno(4))
      arg2(4)=rpic(4)*rpic(1)*(rr(4)**2)+(zz**2)*spicuno(4)*spicdue(4)

      do i = 1,4
      if(abs(arg2(i)).gt.hemips) then
        hsj(i)=atan2(arg1(i),arg2(i))
      else
        hsj(i)=0.0
      end if
      end do
```

```
      vxloc=-(hss(1)*hsq(1)+hss(2)*hsq(2)+hss(3)*hsq(3)+hss(4)*hsq(4))
      vyloc=+(hsc(1)*hsq(1)+hsc(2)*hsq(2)+hsc(3)*hsq(3)+hsc(4)*hsq(4))
      vzloc=+(deteta-hsj(1)-hsj(2)-hsj(3)-hsj(4))

      if(iikk.eq.1) zz = hemips
      if(zz.gt.0.0) then
        vzloc=vzloc
      else
        vzloc=-vzloc
      end if

     vv = rr(1)*hsq(1)+rr(2)*hsq(2)+rr(3)*hsq(3)+rr(4)*hsq(4)+
     $     abs(zz)*(hsj(1)+hsj(2)+hsj(3)+hsj(4)-deteta)
      vvx = vxloc*coscsix + vyloc*cosetax + vzloc*coszitax
      vvy = vxloc*coscsiy + vyloc*cosetay + vzloc*coszitay
      vvz = vxloc*coscsiz + vyloc*cosetaz + vzloc*coszitaz



      return
      end
------------------------------------------------------------------------
------------------------------------------------------------------------
      subroutine splyn(x12,y12,z12,x23,y23,z23,x13,y13,z13,x34,y34,
     $                 z34,x41,y41,z41,xp,yp,zp,w123,w341,f,u,v,w)
------------------------------------------------------------------------

      real*8 x12,y12,z12,x23,y23,z23,x13,y13,z13,x34,y34,
     $       z34,x41,y41,z41,xp,yp,zp,w123,w341,f,u,v,w

      real*8 xj1,yj1,zj1,xj2,yj2,zj2,xj3,yj3,zj3,a1,a2,a3,
     $       den,den1,den2,den3,
     $       fpx1,fpx2,fpx3,
     $       fpy1,fpy2,fpy3,
     $       fpz1,fpz2,fpz3

      xj1 = x23 - xp
      yj1 = y23 - yp
      zj1 = z23 - zp
      xj2 = x12 - xp
      yj2 = y12 - yp
      zj2 = z12 - zp
      xj3 = x13 - xp
      yj3 = y13 - yp
      zj3 = z13 - zp
```

```
a1   = sqrt ( xj1*xj1 + yj1*yj1 + zj1*zj1 )
a2   = sqrt ( xj2*xj2 + yj2*yj2 + zj2*zj2 )
a3   = sqrt ( xj3*xj3 + yj3*yj3 + zj3*zj3 )
den  = 1.0/a1+1.0/a2+1.0/a3
den1 = 1.0/(a1*a1*a1)
den2 = 1.0/(a2*a2*a2)
den3 = 1.0/(a3*a3*a3)

fpx1 = xj1*den1
fpx2 = xj2*den2
fpx3 = xj3*den3
fpy1 = yj1*den1
fpy2 = yj2*den2
fpy3 = yj3*den3
fpz1 = zj1*den1
fpz2 = zj2*den2
fpz3 = zj3*den3

f =   w123 * den
u = - w123 * (fpx1+fpx2+fpx3)
v = - w123 * (fpy1+fpy2+fpy3)
w = - w123 * (fpz1+fpz2+fpz3)

xj1 = x41 - xp
yj1 = y41 - yp
zj1 = z41 - zp
xj2 = x34 - xp
yj2 = y34 - yp
zj2 = z34 - zp
xj3 = x13 - xp
yj3 = y13 - yp
zj3 = z13 - zp

a1   = sqrt ( xj1*xj1 + yj1*yj1 + zj1*zj1 )
a2   = sqrt ( xj2*xj2 + yj2*yj2 + zj2*zj2 )
a3   = sqrt ( xj3*xj3 + yj3*yj3 + zj3*zj3 )
den  = 1.0/a1+1.0/a2+1.0/a3
den1 = 1.0/(a1*a1*a1)
den2 = 1.0/(a2*a2*a2)
den3 = 1.0/(a3*a3*a3)

fpx1 = xj1*den1
fpx2 = xj2*den2
fpx3 = xj3*den3
fpy1 = yj1*den1
fpy2 = yj2*den2
fpy3 = yj3*den3
```

```
      fpz1 = zj1*den1
      fpz2 = zj2*den2
      fpz3 = zj3*den3

      f = f + w341 * den
      u = u - w341 * (fpx1+fpx2+fpx3)
      v = v - w341 * (fpy1+fpy2+fpy3)
      w = w - w341 * (fpz1+fpz2+fpz3)



      return
      end

      subroutine Wave (npan,ym,zm,phase,
     $          nplib,yn,eta0,eta0y,fi0y,fi0z,Am,Wn,Om,hmax,gra,t)

      integer*4 npan,nplib
      real*8    yn(*)
      real*8    ym(*),zm(*)
      real*8    eta0(*),eta0y(*)
      real*8    fi0y(*),fi0z(*)
      real*8    Am,Wn,Om,hmax,gra,t,phase
      integer*4 i

      do i=1,npan
        fi0y(i) = gra*Am*Wn/Om*cosh(Wn*(zm(i)+hmax))/cosh(Wn*hmax)*
     $                              cos(Wn*ym(i)+Om*t+phase)
        fi0z(i) = gra*Am*Wn/Om*sinh(Wn*(zm(i)+hmax))/cosh(Wn*hmax)*
     $                              sin(Wn*ym(i)+Om*t+phase)
      end do
      do i=1,nplib
        eta0(i)  =   - Am*cos(Wn*yn(i)+Om*t+phase)
        eta0y(i) =  Wn*Am*sin(Wn*yn(i)+Om*t+phase)
        fi0y(npan+i) = gra*Wn*Am/Om*cos(Wn*yn(i)+Om*t+phase)
        fi0z(npan+i) = gra*Wn*Am/Om*tanh(Wn*hmax)*
     $                         sin(Wn*yn(i)+Om*t+phase)
      end do



      return
      end
```

```
      subroutine Wave2 (Am,Wn,Om,gra,hmax,t,y,z,phase,fi,u,v,w)
c-----------------------------------------------------------------------
c Calcola le componenti "u,v,w" della velocita in un punto "x,y,z"
c del campo ondoso di caratteristiche "Am,Wn,Om" al tempo "t".
c-----------------------------------------------------------------------
      real*8    y,z,fi,u,v,w
      real*8    Am,Wn,Om,gra,hmax,t,phase
c-----------------------------------------------------------------------
      fi = gra*Am/Om*cosh(Wn*(z+hmax))/cosh(Wn*hmax)*
     $                        sin(Wn*y+Om*t+phase)
       u = 0.d0
       v = gra*Am*Wn/Om*cosh(Wn*(z+hmax))/cosh(Wn*hmax)*
     $                          cos(Wn*y+Om*t+phase)
       w = gra*Am*Wn/Om*sinh(Wn*(z+hmax))/cosh(Wn*hmax)*
     $                          sin(Wn*y+Om*t+phase)


c-----------------------------------------------------------------------
      return
      end
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
      subroutine derita(ntra,nlo,nplib,eta,etax,etay,etasa,etasax,
     $                  etasay,r2dx,r2dy,usa,vsa)

      integer*4 ntra,nlo,nplib,i,j,k
      real*8 r2dx,r2dy,eta(*),etax(*),etay(*),etasa(*),etasax(*),
     $       etasay(*),usa(*),vsa(*),zer,uno,due,tre,qua
      zer = 0.d0
      uno = 1.d0
      due = 2.d0
      tre = 3.d0
      qua = 4.d0

      do i = 1,ntra                                 ! bordo sin !
       if(usa(i).lt.zer) then
         etax(i) = due*(eta(i+ntra)-eta(i))*r2dx
         etasax(i) = due*(etasa(i+ntra)-etasa(i))*r2dx
       else
         etax(i) = zer
         etasax(i) = zer
       end if
      end do
```

```
      do i = nplib-ntra+1,nplib                    ! bordo dx !
        if(usa(i).ge.zer) then
          etax(i) = due*(eta(i)-eta(i-ntra))*r2dx
          etasax(i) = due*(etasa(i)-etasa(i-ntra))*r2dx
        else
          etax(i) = zer
          etasax(i) = zer
        end if
      end do
      do i = ntra+1,2*ntra                         ! sub-bordo sin !
        if(usa(i).ge.zer) then
          etax(i) = due*(eta(i)-eta(i-ntra))*r2dx
          etasax(i) = due*(etasa(i)-etasa(i-ntra))*r2dx
        else
          etax(i) = (-tre*eta(i)+qua*eta(i+ntra)-eta(i+2*ntra))*r2dx
          etasax(i) = (-tre*etasa(i)+qua*etasa(i+ntra)
     $                          -etasa(i+2*ntra))*r2dx
        end if
      end do
      do i = nplib-2*ntra+1,nplib-ntra             ! sub-bordo dx !
        if(usa(i).ge.zer) then
          etax(i) = (tre*eta(i)-qua*eta(i-ntra)+eta(i-2*ntra))*r2dx
          etasax(i) = (tre*etasa(i)-qua*etasa(i-ntra)
     $                          +etasa(i-2*ntra))*r2dx
        else
          etax(i) = due*(eta(i+ntra)-eta(i))*r2dx
          etasax(i) = due*(etasa(i+ntra)-etasa(i))*r2dx
        end if
      end do
      do i = 2*ntra+1,nplib-2*ntra                 ! middle !
        if(usa(i).ge.zer) then
          etax(i) = (tre*eta(i)-qua*eta(i-ntra)+eta(i-2*ntra))*r2dx
          etasax(i) = (tre*etasa(i)-qua*etasa(i-ntra)
     $                          +etasa(i-2*ntra))*r2dx
        else
          etax(i) = (-tre*eta(i)+qua*eta(i+ntra)-eta(i+2*ntra))*r2dx
          etasax(i) = (-tre*etasa(i)+qua*etasa(i+ntra)
     $                          -etasa(i+2*ntra))*r2dx
        end if
      end do

      do i = 1,nplib-ntra+1,ntra                   ! bordo sud !
        if(vsa(i).ge.zer) then
          etay(i) = zer
          etasay(i) = zer
        else
          etay(i) = due*(eta(i+1)-eta(i))*r2dy
```

```
    etasay(i) = due*(etasa(i+1)-etasa(i))*r2dy
   end if
  end do
  do i = ntra,nplib,ntra                    ! bordo nord !
   if(vsa(i).ge.zer) then
    etay(i) = due*(eta(i)-eta(i-1))*r2dy
    etasay(i) = due*(etasa(i)-etasa(i-1))*r2dy
   else
    etay(i) = zer
    etasay(i) = zer
   end if
  end do
  do i = 2,nplib-ntra+2,ntra                ! sub-bordo sud !
   if(vsa(i).ge.zer) then
    etay(i) = due*(eta(i)-eta(i-1))*r2dy
    etasay(i) = due*(etasa(i)-etasa(i-1))*r2dy
   else
    etay(i) = (-tre*eta(i)+qua*eta(i+1)-eta(i+2))*r2dy
    etasay(i) = (-tre*etasa(i)+qua*etasa(i+1)-etasa(i+2))*r2dy
   end if
  end do
  do i = ntra-1,nplib-1,ntra                ! sub-bordo nord !
   if(vsa(i).ge.zer) then
    etay(i) = (tre*eta(i)-qua*eta(i-1)+eta(i-2))*r2dy
    etasay(i) = (tre*etasa(i)-qua*etasa(i-1)+etasa(i-2))*r2dy
   else
    etay(i) = due*(eta(i+1)-eta(i))*r2dy
    etasay(i) = due*(etasa(i+1)-etasa(i))*r2dy
   end if
  end do
  do i = 1,nlo                              ! middle !
   do j = 3,ntra-2
    k = (i-1)*ntra + j
    if(vsa(k).ge.zer) then
     etay(k) = (tre*eta(k)-qua*eta(k-1)+eta(k-2))*r2dy
     etasay(k) = (tre*etasa(k)-qua*etasa(k-1)+etasa(k-2))*r2dy
    else
     etay(k) = (-tre*eta(k)+qua*eta(k+1)-eta(k+2))*r2dy
     etasay(k) = (-tre*etasa(k)+qua*etasa(k+1)-etasa(k+2))*r2dy
    end if
   end do
  end do


  return
  end
```

```
      subroutine rkiniz (ikutta,rk)
c_____________________________________________________________________
c
c Inizializza il vettore rk delle costanti di integrazione numerica
c ikutta = ordine schema di integrazione
c_____________________________________________________________________

      integer*4 ikutta
      real*8    rk(1:4,1:2)
_____________________________________________________________________

 Eulero
      if (ikutta .eq. 1) then
         rk(1,1) = 1.0
         rk(1,2) = 0.0
      end if

 R-K secondo ordine
      if (ikutta .eq. 2) then
        rk(1,1) = .50
        rk(2,1) = .50
        rk(1,2) = 0.0
        rk(2,2) = 1.0
      end if

 R-K quarto ordine
      if (ikutta .eq. 4) then
        rk(1,1) = 1.0 / 6.0
        rk(2,1) = 1.0 / 3.0
        rk(3,1) = 1.0 / 3.0
        rk(4,1) = 1.0 / 6.0
        rk(1,2) = 0.0
        rk(2,2) = .50
        rk(3,2) = .50
        rk(4,2) = 1.0
      end if




_____________________________________________________________________
      return
      end
_____________________________________________________________________
—

_____________________________________________________________
```

```
      subroutine rukut1 (nplib,etan,fin,eta,teta,fi,tofi,rk,ik,dt)
_____________________________________________________________________

      integer*4 nplib,i
      real*8    etan(nplib)  ,eta(nplib)
      real*8    fin(nplib)   ,fi(nplib)
      real*8    tofi(nplib)  ,teta(nplib)

      integer*4 ik
      real*8    dt
      real*8    rk(1:4,1:2)
_____________________________________________________________________

      do 100 i=1,nplib
        fin(i)  =  fi(i) + dt*rk(ik,2) * tofi(i)
        etan(i) = eta(i) + dt*rk(ik,2) * teta(i)
        tofi(i) = 0.0
        teta(i) = 0.0
100   continue



_____________________________________________________________________
      return
      end
_____________________________________________________________________
_____________________________________________________________________
      subroutine rukut2(npan,nplib,deta,dfi,etan,fin,fix,fiy,fiz,
     $              etax,etay,fi0y,eta0y,usa,vsa,disa,
     $             etasax,etasay,spinu,tofi,teta,gra,rk,ik)


      integer*4 npan,nplib,i,ik
      real*8 gra,deta(*),dfi(*),etan(*),fin(*),fix(*),
     $ fiy(*),fiz(*),etax(*),etay(*),fi0y(*),eta0y(*),tofi(*),
     $ teta(nplib),spinu(nplib),rk(1:4,1:2)
      real*8 usa(*),vsa(*),etasax(*),etasay(*),disa(*)


      do 50 i=1,nplib
        tofi(i) = - gra*etan(i) - usa(i)*fix(npan+i)
     $                        - vsa(i)*(fi0y(npan+i)+fiy(npan+i))
     $            - spinu(i)*fin(i)      ! careful manca integrale vor !
        teta(i) = fiz(npan+i) + disa(i) - usa(i)*etax(i)
     $                                - vsa(i)*(eta0y(i)+etay(i))
     $                           - fix(npan+i)*etasax(i)
     $             - (fi0y(npan+i)+fiy(npan+i))*etasay(i)
     $             - spinu(i)*etan(i)
```

```
50    continue

      do 100 i=1,nplib
        deta(i) = deta(i) + rk(ik,1) * teta(i)
        dfi(i)  = dfi(i)  + rk(ik,1) * tofi(i)
100   continue

      return
      end
_____________________________________________________________________
_____________________________________________________________________
      subroutine rukut3 (nplib,eta,fi,deta,dfi,dt)
_____________________________________________________________________

      integer*4 nplib
      real*8 dt
      real*8 eta(nplib),fi(nplib)
      real*8 deta(nplib),dfi(nplib)
      integer*4 i
_____________________________________________________________________

      do 100 i=1,nplib
        eta(i) = eta(i) + dt * deta(i)
        fi(i)  = fi(i)  + dt * dfi(i)
        deta(i) = 0.0
        dfi(i)  = 0.0
100   continue
_____________________________________________________________________
      return
      end
**********************************************************************
      subroutine dgeco(a,lda,n,ipvt,rcond,z)
      integer lda,n,ipvt(1)
      double precision a(lda,1),z(1)
      double precision rcond

     dgeco factors a double precision matrix by gaussian elimination
     and estimates the condition of the matrix.

     if  rcond  is not needed, dgefa is slightly faster.
    to solve  a*x = b , follow dgeco by dgesl.
     to compute  inverse(a)*c , follow dgeco by dgesl.
     to compute  determinant(a) , follow dgeco by dgedi.
    to compute  inverse(a) , follow dgeco by dgedi.

     on entry
```

```
    a       double precision(lda, n)
            the matrix to be factored.

    lda     integer
            the leading dimension of the array  a .

    n       integer
            the order of the matrix  a .

  on return

    a       an upper triangular matrix and the multipliers
            which were used to obtain it.
            the factorization can be written  a = l*u  where
            l  is a product of permutation and unit lower
            triangular matrices and  u  is upper triangular.

    ipvt    integer(n)
            an integer vector of pivot indices.

    rcond   double precision
            an estimate of the reciprocal condition of  a .
            for the system  a*x = b , relative perturbations
            in  a  and  b  of size  epsilon  may cause
            relative perturbations in  x  of size  epsilon/rcond .
            if  rcond  is so small that the logical expression
                    1.0 + rcond .eq. 1.0
            is true, then  a  may be singular to working
            precision.  in particular,  rcond  is zero  if
            exact singularity is detected or the estimate
            underflows.

    z       double precision(n)
            a work vector whose contents are usually unimportant.
            if  a  is close to a singular matrix, then  z  is
            an approximate null vector in the sense that
            norm(a*z) = rcond*norm(a)*norm(z) .

  linpack. this version dated 08/14/78 .
  cleve moler, university of new mexico, argonne national lab.

  subroutines and functions

  linpack dgefa
  blas daxpy,ddot,dscal,dasum
  fortran dabs,dmax1,dsign
  internal variables
```

```
      double precision ddot,ek,t,wk,wkm
      double precision anorm,s,dasum,sm,ynorm
      integer info,j,k,kb,kp1,l


      compute 1-norm of a

      anorm = 0.0d0
      do 10 j = 1, n
         anorm = dmax1(anorm,dasum(n,a(1,j),1))
   10 continue

      factor

      call dgefa(a,lda,n,ipvt,info)

      rcond = 1/(norm(a)*(estimate of norm(inverse(a)))) .
      estimate = norm(z)/norm(y) where  a*z = y  and  trans(a)*y = e .
      trans(a)  is the transpose of a .  the components of  e  are
      chosen to cause maximum local growth in the elements of w  where
      trans(u)*w = e .  the vectors are frequently rescaled to avoid
      overflow.

      solve trans(u)*w = e

      ek = 1.0d0
      do 20 j = 1, n
         z(j) = 0.0d0
   20 continue
      do 100 k = 1, n
         if (z(k) .ne. 0.0d0) ek = dsign(ek,-z(k))
         if (dabs(ek-z(k)) .le. dabs(a(k,k))) go to 30
            s = dabs(a(k,k))/dabs(ek-z(k))
            call dscal(n,s,z,1)
            ek = s*ek
   30    continue
         wk = ek - z(k)
         wkm = -ek - z(k)
         s = dabs(wk)
         sm = dabs(wkm)
         if (a(k,k) .eq. 0.0d0) go to 40
            wk = wk/a(k,k)
            wkm = wkm/a(k,k)
         go to 50
   40    continue
            wk = 1.0d0
```

```
         wkm = 1.0d0
 50    continue
       kp1 = k + 1
       if (kp1 .gt. n) go to 90
          do 60 j = kp1, n
             sm = sm + dabs(z(j)+wkm*a(k,j))
             z(j) = z(j) + wk*a(k,j)
             s = s + dabs(z(j))
 60       continue
          if (s .ge. sm) go to 80
             t = wkm - wk
             wk = wkm
             do 70 j = kp1, n
                z(j) = z(j) + t*a(k,j)
 70          continue
 80       continue
 90    continue
       z(k) = wk
100 continue
    s = 1.0d0/dasum(n,z,1)
    call dscal(n,s,z,1)

   solve trans(l)*y = w

   do 120 kb = 1, n
      k = n + 1 - kb
      if (k .lt. n) z(k) = z(k) + ddot(n-k,a(k+1,k),1,z(k+1),1)
      if (dabs(z(k)) .le. 1.0d0) go to 110
         s = 1.0d0/dabs(z(k))
         call dscal(n,s,z,1)
110    continue
       l = ipvt(k)
       t = z(l)
       z(l) = z(k)
       z(k) = t
120 continue
    s = 1.0d0/dasum(n,z,1)
    call dscal(n,s,z,1)

   ynorm = 1.0d0

   solve l*v = y

    do 140 k = 1, n
       l = ipvt(k)
       t = z(l)
       z(l) = z(k)
```

```
      z(k) = t
      if (k .lt. n) call daxpy(n-k,t,a(k+1,k),1,z(k+1),1)
      if (dabs(z(k)) .le. 1.0d0) go to 130
        s = 1.0d0/dabs(z(k))
        call dscal(n,s,z,1)
        ynorm = s*ynorm
130   continue
140 continue
    s = 1.0d0/dasum(n,z,1)
    call dscal(n,s,z,1)
    ynorm = s*ynorm

   solve  u*z = v

    do 160 kb = 1, n
      k = n + 1 - kb
      if (dabs(z(k)) .le. dabs(a(k,k))) go to 150
        s = dabs(a(k,k))/dabs(z(k))
        call dscal(n,s,z,1)
        ynorm = s*ynorm
150   continue
      if (a(k,k) .ne. 0.0d0) z(k) = z(k)/a(k,k)
      if (a(k,k) .eq. 0.0d0) z(k) = 1.0d0
      t = -z(k)
      call daxpy(k-1,t,a(1,k),1,z(1),1)
160 continue
   make znorm = 1.0
    s = 1.0d0/dasum(n,z,1)
    call dscal(n,s,z,1)
    ynorm = s*ynorm

    if (anorm .ne. 0.0d0) rcond = ynorm/anorm
    if (anorm .eq. 0.0d0) rcond = 0.0d0
    return
    end
*****************************************************************************
    subroutine dgesl(a,lda,n,ipvt,b,job)
    integer lda,n,ipvt(1),job
    double precision a(lda,1),b(1)

   dgesl solves the double precision system
   a * x = b  or  trans(a) * x = b
   using the factors computed by dgeco or dgefa.

   on entry

      a       double precision(lda, n)
```

```
          the output from dgeco or dgefa.

   lda    integer
          the leading dimension of the array  a .

   n      integer
          the order of the matrix  a .

   ipvt   integer(n)
          the pivot vector from dgeco or dgefa.

   b      double precision(n)
          the right hand side vector.

   job    integer
          = 0         to solve  a*x = b ,
          = nonzero   to solve  trans(a)*x = b  where
                      trans(a)  is the transpose.

on return

   b      the solution vector  x .

error condition

   a division by zero will occur if the input factor contains a
   zero on the diagonal.  technically this indicates singularity
   but it is often caused by improper arguments or improper
   setting of lda .  it will not occur if the subroutines are
   called correctly and if dgeco has set rcond .gt. 0.0
   or dgefa has set info .eq. 0 .

to compute  inverse(a) * c  where  c  is a matrix
with  p  columns
      call dgeco(a,lda,n,ipvt,rcond,z)
      if (rcond is too small) go to ...
      do 10 j = 1, p
         call dgesl(a,lda,n,ipvt,c(1,j),0)
   10 continue

linpack. this version dated 08/14/78 .
cleve moler, university of new mexico, argonne national lab.

subroutines and functions
blas daxpy,ddot

internal variables
```

```
      double precision ddot,t
      integer k,kb,l,nm1

      nm1 = n - 1
      if (job .ne. 0) go to 50

        job = 0 , solve  a * x = b
        first solve  l*y = b

        if (nm1 .lt. 1) go to 30
         do 20 k = 1, nm1
            l = ipvt(k)
            t = b(l)
            if (l .eq. k) go to 10
               b(l) = b(k)
               b(k) = t
10          continue
            call daxpy(n-k,t,a(k+1,k),1,b(k+1),1)
20       continue
30       continue

        now solve  u*x = y

         do 40 kb = 1, n
            k = n + 1 - kb
            b(k) = b(k)/a(k,k)
            t = -b(k)
            call daxpy(k-1,t,a(1,k),1,b(1),1)
40       continue
      go to 100
50    continue

        job = nonzero, solve  trans(a) * x = b
        first solve  trans(u)*y = b

         do 60 k = 1, n
            t = ddot(k-1,a(1,k),1,b(1),1)
            b(k) = (b(k) - t)/a(k,k)
60       continue

        now solve trans(l)*x = y

         if (nm1 .lt. 1) go to 90
         do 80 kb = 1, nm1
            k = n - kb
            b(k) = b(k) + ddot(n-k,a(k+1,k),1,b(k+1),1)
```

```
         l = ipvt(k)
         if (l .eq. k) go to 70
            t = b(l)
            b(l) = b(k)
            b(k) = t
   70    continue
   80  continue
   90  continue
  100 continue
      return
      end
*****************************************************************************
      subroutine dgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(1),info
      double precision a(lda,1)

    dgefa factors a double precision matrix by gaussian elimination.

    dgefa is usually called by dgeco, but it can be called
    directly with a saving in time if  rcond  is not needed.
    (time for dgeco) = (1 + 9/n)*(time for dgefa) .

    on entry

       a       double precision(lda, n)
               the matrix to be factored.

       lda     integer
               the leading dimension of the array  a .

       n       integer
               the order of the matrix  a .
    on return

       a       an upper triangular matrix and the multipliers
               which were used to obtain it.
               the factorization can be written  a = l*u  where
               l  is a product of permutation and unit lower
               triangular matrices and  u  is upper triangular.

       ipvt    integer(n)
              an integer vector of pivot indices.

      info    integer
               = 0  normal value.
               = k  if  u(k,k) .eq. 0.0 .  this is not an error
                    condition for this subroutine, but it does
```

```
              indicate that dgesl or dgedi will divide by zero
              if called.  use  rcond  in dgeco for a reliable
              indication of singularity.

   linpack. this version dated 08/14/78 .
   cleve moler, university of new mexico, argonne national lab.
   subroutines and functions

   blas daxpy,dscal,idamax

   internal variables

    double precision t
    integer idamax,j,k,kp1,l,nm1


   gaussian elimination with partial pivoting

    info = 0
    nm1 = n - 1
    if (nm1 .lt. 1) go to 70
    do 60 k = 1, nm1
       kp1 = k + 1

      find l = pivot index

       l = idamax(n-k+1,a(k,k),1) + k - 1
       ipvt(k) = l

      zero pivot implies this column already triangularized
       if (a(l,k) .eq. 0.0d0) go to 40

         interchange if necessary

          if (l .eq. k) go to 10
             t = a(l,k)
             a(l,k) = a(k,k)
             a(k,k) = t
10        continue

         compute multipliers

          t = -1.0d0/a(k,k)
          call dscal(n-k,t,a(k+1,k),1)

         row elimination with column indexing
```

```
         do 30 j = kp1, n
            t = a(l,j)
            if (l .eq. k) go to 20
               a(l,j) = a(k,j)
               a(k,j) = t
   20       continue
            call daxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
   30    continue
      go to 50
   40 continue
         info = k
   50 continue
   60 continue
   70 continue
     ipvt(n) = n
     if (a(n,n) .eq. 0.0d0) info = n
     return
     end
*****************************************************************************
     subroutine  dscal(n,da,dx,incx)

    scales a vector by a constant.
    uses unrolled loops for increment equal to one.
    jack dongarra, linpack, 3/11/78.
    modified 3/93 to return if incx .le. 0.
    modified 12/3/93, array(1) declarations changed to array(*)

     double precision da,dx(*)
     integer i,incx,m,mp1,n,nincx

    if( n.le.0 .or. incx.le.0 )return
    if(incx.eq.1)go to 20

      code for increment not equal to 1
     nincx = n*incx
     do 10 i = 1,nincx,incx
       dx(i) = da*dx(i)
   10 continue
     return

      code for increment equal to 1


      clean-up loop

   20 m = mod(n,5)
     if( m .eq. 0 ) go to 40
```

```
      do 30 i = 1,m
        dx(i) = da*dx(i)
   30 continue
      if( n .lt. 5 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dx(i) = da*dx(i)
        dx(i + 1) = da*dx(i + 1)
        dx(i + 2) = da*dx(i + 2)
        dx(i + 3) = da*dx(i + 3)
        dx(i + 4) = da*dx(i + 4)
   50 continue
      return
      end
*************************************************************************
      subroutine daxpy(n,da,dx,incx,dy,incy)

     constant times a vector plus a vector.
     uses unrolled loops for increments equal to one.
     jack dongarra, linpack, 3/11/78.
     modified 12/3/93, array(1) declarations changed to array(*)

     double precision dx(*),dy(*),da
      integer i,incx,incy,ix,iy,m,mp1,n

      if(n.le.0)return
      if (da .eq. 0.0d0) return
      if(incx.eq.1.and.incy.eq.1)go to 20

        code for unequal increments or equal increments
          not equal to 1

      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return

        code for both increments equal to 1


        clean-up loop
```

```
 20 m = mod(n,4)
    if( m .eq. 0 ) go to 40
    do 30 i = 1,m
      dy(i) = dy(i) + da*dx(i)
 30 continue
    if( n .lt. 4 ) return
 40 mp1 = m + 1
    do 50 i = mp1,n,4
      dy(i) = dy(i) + da*dx(i)
      dy(i + 1) = dy(i + 1) + da*dx(i + 1)
      dy(i + 2) = dy(i + 2) + da*dx(i + 2)
      dy(i + 3) = dy(i + 3) + da*dx(i + 3)
 50 continue
    return
    end
*********************************************************************
    double precision function dasum(n,dx,incx)

   takes the sum of the absolute values.
   jack dongarra, linpack, 3/11/78.
   modified 3/93 to return if incx .le. 0.
   modified 12/3/93, array(1) declarations changed to array(*)

    double precision dx(*),dtemp
    integer i,incx,m,mp1,n,nincx

    dasum = 0.0d0
    dtemp = 0.0d0
    if( n.le.0 .or. incx.le.0 )return
    if(incx.eq.1)go to 20

      code for increment not equal to 1

   nincx = n*incx
    do 10 i = 1,nincx,incx
      dtemp = dtemp + dabs(dx(i))
 10 continue
    dasum = dtemp
    return

      code for increment equal to 1


      clean-up loop

 20 m = mod(n,6)
```

```
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dabs(dx(i))
   30 continue
      if( n .lt. 6 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,6
        dtemp = dtemp + dabs(dx(i)) + dabs(dx(i + 1)) + dabs(dx(i + 2))
     *  + dabs(dx(i + 3)) + dabs(dx(i + 4)) + dabs(dx(i + 5))
   50 continue
   60 dasum = dtemp
      return
      end
*************************************************************************
      double precision function ddot(n,dx,incx,dy,incy)

     forms the dot product of two vectors.
     uses unrolled loops for increments equal to one.
     jack dongarra, linpack, 3/11/78.
     modified 12/3/93, array(1) declarations changed to array(*)

      double precision dx(*),dy(*),dtemp
     integer i,incx,incy,ix,iy,m,mp1,n

     ddot = 0.0d0
      dtemp = 0.0d0
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20

        code for unequal increments or equal increments
          not equal to 1

      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dtemp + dx(ix)*dy(iy)
        ix = ix + incx
        iy = iy + incy
   10 continue
      ddot = dtemp
      return

        code for both increments equal to 1
```

```
      clean-up loop

   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dx(i)*dy(i)
   30 continue
      if( n .lt. 5 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dtemp = dtemp + dx(i)*dy(i) + dx(i + 1)*dy(i + 1) +
     *   dx(i + 2)*dy(i + 2) + dx(i + 3)*dy(i + 3) + dx(i + 4)*dy(i + 4)
   50 continue
   60 ddot = dtemp
      return
      end
*********************************************************************
      integer function idamax(n,dx,incx)

     finds the index of element having max. absolute value.
     jack dongarra, linpack, 3/11/78.
     modified 3/93 to return if incx .le. 0.
     modified 12/3/93, array(1) declarations changed to array(*)

      double precision dx(*),dmax
      integer i,incx,ix,n

      idamax = 0
      if( n.lt.1 .or. incx.le.0 ) return
      idamax = 1
      if(n.eq.1)return
      if(incx.eq.1)go to 20

        code for increment not equal to 1

      ix = 1
      dmax = dabs(dx(1))
      ix = ix + incx
      do 10 i = 2,n
        if(dabs(dx(ix)).le.dmax) go to 5
        idamax = i
        dmax = dabs(dx(ix))
    5   ix = ix + incx
   10 continue
      return

        code for increment equal to 1
```

```
 20 dmax = dabs(dx(1))
    do 30 i = 2,n
       if(dabs(dx(i)).le.dmax) go to 30
       idamax = i
       dmax = dabs(dx(i))
 30 continue
    return
    end
*****************************************************************************
    subroutine etano(pres,preplo,eta0,mx,my,dx,dy,noradi)
    integer mx,mx1,mx2,my,my1,my2,i,j,noradi
    real*8 pres(mx,my),preplo(mx,my),dx(mx),dy(my)
    real*8 eta0,dist

    mx1 = mx-1
    mx2 = mx-2
    my1 = my-1
    my2 = my-2

    preplo(1,1) = pres(1,1)
    preplo(mx,1) = pres(mx1,1)
    preplo(1,my) = eta0*float(noradi)+float(1-noradi)*pres(1,my1)
    preplo(mx,my) = eta0*float(noradi)+float(1-noradi)*pres(mx1,my1)
    do i = 2,mx1
      dist = dx(i)+dx(i-1)
      preplo(i,1) = (pres(i,1)*dx(i-1)+pres(i-1,1)*dx(i))/dist
      preplo(i,my) = float(noradi)*eta0 + float(1-noradi)*
   $              (pres(i,my1)*dx(i-1)+pres(i-1,my1)*dx(i))/dist
    end do
    do j = 2,my1
      dist = dy(j)+dy(j-1)
      preplo(1,j) = (pres(1,j)*dy(j-1)+pres(1,j-1)*dy(j))/dist
      preplo(mx,j) = (pres(mx1,j)*dy(j-1)+pres(mx1,j-1)*dy(j))/dist
    end do

    do j = 2,my1
      do i = 2,mx1
        dist = (dx(i)+dx(i-1))*(dy(j)+dy(j-1))
        preplo(i,j) = ( pres(i,j)*dx(i-1)*dy(j-1)
   $                 +pres(i-1,j)*dx(i)*dy(j-1)
   $                 +pres(i,j-1)*dx(i-1)*dy(j)
   $                 +pres(i-1,j-1)*dx(i)*dy(j) )/dist
      end do
    end do

    return
```

```
      end
**************************************************************************
      subroutine capre(pres,preplo,prof,mx,my,dx,dy)
      integer mx,mx1,mx2,my,my1,my2,i,j
      real*8 pres(mx,my),preplo(mx,my),prof(mx,my),dx(mx),dy(my)
      real*8 dist

      mx1 = mx-1
      mx2 = mx-2
      my1 = my-1
      my2 = my-2

      preplo(1,1) = pres(1,1)/prof(1,1)
      preplo(mx,1) = pres(mx1,1)/prof(mx1,1)
      preplo(1,my) = pres(1,my1)/prof(1,my1)
      preplo(mx,my) = pres(mx1,my1)/prof(mx1,my1)
      do i = 2,mx1
        dist = dx(i)+dx(i-1)
        preplo(i,1) = (pres(i,1)/prof(i,1)*dx(i-1)+
     $               pres(i-1,1)/prof(i-1,1)*dx(i))/dist
        preplo(i,my) = (pres(i,my1)/prof(i,my1)*dx(i-1)+
     $                pres(i-1,my1)/prof(i-1,my1)*dx(i))/dist
      end do
      do j = 2,my1
        dist = dy(j)+dy(j-1)
        preplo(1,j) = (pres(1,j)/prof(1,j)*dy(j-1)+
     $               pres(1,j-1)/prof(1,j-1)*dy(j))/dist
        preplo(mx,j) = (pres(mx1,j)/prof(mx1,j)*dy(j-1)+
     $                pres(mx1,j-1)/prof(mx1,j-1)*dy(j))/dist
      end do

      do j = 2,my1
        do i = 2,mx1
          dist = (dx(i)+dx(i-1))*(dy(j)+dy(j-1))
          preplo(i,j) = (pres(i,j)/prof(i,j)*dx(i-1)*dy(j-1)
     $                 +pres(i-1,j)/prof(i-1,j)*dx(i)*dy(j-1)
     $                 +pres(i,j-1)/prof(i,j-1)*dx(i-1)*dy(j)
     $                 +pres(i-1,j-1)/prof(i-1,j-1)*dx(i)*dy(j))/dist
        end do
      end do

      return
      end



**************************************************************************
```

```
      subroutine casiplo(siplo,u,v,mx,mx1,my,my1,dx,dy)
      integer mx,my,mx1,my1,i,j
      real*8 u(mx,my),v(mx,my),siplo(mx,my),dx(mx),dy(my),
     $     zer,uqt,umz,uno,due,tre,qua

      zer = 0.0d0
      uqt = 0.25d0
      umz = 0.5d0
      uno = 1.0d0
      due = 2.0d0
      tre = 3.0d0
      qua = 4.0d0


       i = 1                                ! ang. inf. sn. !
       j = 1
       siplo(i,j) =
     $       ((-tre*u(i,j)+qua*u(i+1,j)-u(i+2,j))/(due*dx(i)))**2
     $     + ((-tre*v(i,j)+qua*v(i,j+1)-v(i,j+2))/(due*dy(j)))**2
     $  + umz*( (-tre*u(i,j)+qua*u(i,j+1)-u(i,j+2))/(due*dy(j))+
     $        (-tre*v(i,j)+qua*v(i+1,j)-v(i+2,j))/(due*dx(i))
     $      )**2
       siplo(i,j) = sqrt(siplo(i,j))

       i = mx                               ! ang. inf. dx. !
       j = 1
       siplo(i,j) =
     $       ((tre*u(i,j)-qua*u(i-1,j)+u(i-2,j))/(due*dx(i-1)))**2
     $     + ((-tre*v(i,j)+qua*v(i,j+1)-v(i,j+2))/(due*dy(j)))**2
     $  + umz*( (-tre*u(i,j)+qua*u(i,j+1)-u(i,j+2))/(due*dy(j))+
     $        (tre*v(i,j)-qua*v(i-1,j)+v(i-2,j))/(due*dx(i-1))
     $      )**2
       siplo(i,j) = sqrt(siplo(i,j))


      j = 1                                 ! b. inf. !
      do i = 2,mx1
       siplo(i,j) =
     $     ((dx(i-1)**2*u(i+1,j)+(dx(i)**2-dx(i-1)**2)*u(i,j)
     $             -dx(i)**2*u(i-1,j))/
     $        (dx(i-1)*dx(i)*(dx(i-1)+dx(i))))**2 +
     $      ((-tre*v(i,j)+qua*v(i,j+1)-v(i,j+2))/(due*dy(j)))**2
     $  + umz*( (-tre*u(i,j)+qua*u(i,j+1)-u(i,j+2))/(due*dy(j))+
     $      (dx(i-1)**2*v(i+1,j)+(dx(i)**2-dx(i-1)**2)*v(i,j)
     $             -dx(i)**2*v(i-1,j))/
     $        (dx(i-1)*dx(i)*(dx(i-1)+dx(i)))
     $      )**2
```

```
      siplo(i,j) = sqrt(siplo(i,j))
    end do

    do j = 2,my1                              ! parte centrale !
      do i = 2,mx1
        siplo(i,j) =
     $       ((dx(i-1)**2*u(i+1,j)+(dx(i)**2-dx(i-1)**2)*u(i,j)
     $                 -dx(i)**2*u(i-1,j))/
     $            (dx(i-1)*dx(i)*(dx(i-1)+dx(i))))**2 +
     $       ((dy(j-1)**2*v(i,j+1)+(dy(j)**2-dy(j-1)**2)*v(i,j)
     $                 -dy(j)**2*v(i,j-1))/
     $            (dy(j-1)*dy(j)*(dy(j-1)+dy(j))))**2 +
     $   + umz*(
     $        (dy(j-1)**2*u(i,j+1)+(dy(j)**2-dy(j-1)**2)*u(i,j)
     $                 -dy(j)**2*u(i,j-1))/
     $            (dy(j-1)*dy(j)*(dy(j-1)+dy(j))) +
     $        (dx(i-1)**2*v(i+1,j)+(dx(i)**2-dx(i-1)**2)*v(i,j)
     $                 -dx(i)**2*v(i-1,j))/
     $            (dx(i-1)*dx(i)*(dx(i-1)+dx(i)))
     $        )**2
        siplo(i,j) = sqrt(siplo(i,j))
      end do
    end do

      i = 1                                   ! ang. sup. sn. !
      j = my
      siplo(i,j) =
     $         ((-tre*u(i,j)+qua*u(i+1,j)-u(i+2,j))/(due*dx(i)))**2
     $       + ((tre*v(i,j)-qua*v(i,j-1)+v(i,j-2))/(due*dy(j-1)))**2
     $   + umz*( (tre*u(i,j)-qua*u(i,j-1)+u(i,j-2))/(due*dy(j-1))+
     $           (-tre*v(i,j)+qua*v(i+1,j)-v(i+2,j))/(due*dx(i))
     $         )**2
      siplo(i,j) = sqrt(siplo(i,j))

      i = mx                                  ! ang. sup. dx. !
      j = my
      siplo(i,j) =
     $         ((tre*u(i,j)-qua*u(i-1,j)+u(i-2,j))/(due*dx(i-1)))**2
     $       + ((tre*v(i,j)-qua*v(i,j-1)+v(i,j-2))/(due*dy(j-1)))**2
     $   + umz*( (tre*u(i,j)-qua*u(i,j-1)+u(i,j-2))/(due*dy(j-1))+
     $           (tre*v(i,j)-qua*v(i-1,j)+v(i-2,j))/(due*dx(i-1))
     $         )**2
      siplo(i,j) = sqrt(siplo(i,j))


    j = my                                    ! b. sup. !
    do i = 2,mx1
```

```
      siplo(i,j) =
     $       ((dx(i-1)**2*u(i+1,j)+(dx(i)**2-dx(i-1)**2)*u(i,j)
     $                -dx(i)**2*u(i-1,j))/
     $           (dx(i-1)*dx(i)*(dx(i-1)+dx(i))))**2 +
     $        ((tre*v(i,j)-qua*v(i,j-1)+v(i,j-2))/(due*dy(j-1)))**2
     $   + umz*( (tre*u(i,j)-qua*u(i,j-1)+u(i,j-2))/(due*dy(j-1))+
     $       (dx(i-1)**2*v(i+1,j)+(dx(i)**2-dx(i-1)**2)*v(i,j)
     $                -dx(i)**2*v(i-1,j))/
     $           (dx(i-1)*dx(i)*(dx(i-1)+dx(i)))
     $       )**2
      siplo(i,j) = sqrt(siplo(i,j))
     end do


     i = 1                                          ! b. sn. !
     do j = 2,my1
      siplo(i,j) =
     $       ((-tre*u(i,j)+qua*u(i+1,j)-u(i+2,j))/(due*dx(i)))**2
     $     + ((dy(j-1)**2*v(i,j+1)+(dy(j)**2-dy(j-1)**2)*v(i,j)
     $                -dy(j)**2*v(i,j-1))/
     $           (dy(j-1)*dy(j)*(dy(j-1)+dy(j))))**2 +
     $   + umz*(
     $       (dy(j-1)**2*u(i,j+1)+(dy(j)**2-dy(j-1)**2)*u(i,j)
     $                -dy(j)**2*u(i,j-1))/
     $           (dy(j-1)*dy(j)*(dy(j-1)+dy(j))) +
     $         (-tre*v(i,j)+qua*v(i+1,j)-v(i+2,j))/(due*dx(i))
     $       )**2
      siplo(i,j) = sqrt(siplo(i,j))
     end do

     i = mx                                           ! b. dx. !
     do j = 2,my1
      siplo(i,j) =
     $       ((tre*u(i,j)-qua*u(i-1,j)+u(i-2,j))/(due*dx(i-1)))**2
     $     + ((dy(j-1)**2*v(i,j+1)+(dy(j)**2-dy(j-1)**2)*v(i,j)
     $                -dy(j)**2*v(i,j-1))/
     $           (dy(j-1)*dy(j)*(dy(j-1)+dy(j))))**2 +
     $   + umz*(
     $       (dy(j-1)**2*u(i,j+1)+(dy(j)**2-dy(j-1)**2)*u(i,j)
     $                -dy(j)**2*u(i,j-1))/
     $           (dy(j-1)*dy(j)*(dy(j-1)+dy(j))) +
     $         (tre*v(i,j)-qua*v(i-1,j)+v(i-2,j))/(due*dx(i-1))
     $       )**2
      siplo(i,j) = sqrt(siplo(i,j))
     end do
```

```
      return
      end
*************************************************************************
      subroutine unoli(Hu,u,v,dx,dy,prof,proplo,smart,mx,mx1,mx2,my,
     $           my1,my2,istre_w,istre_e,jstre,ider_w,ider_e,ider_n)
      integer mx,mx1,mx2,my,my1,my2,istre_w,istre_e,jstre,ider,
     $ ider_w,ider_e,ider_n,i,j,k
      real*8 u(mx,0:my),v(0:mx,my),Hu(2,mx,my),prof(mx,my),smart,
     $ proplo(mx,my),dx(*),dy(*),hij,hjp,hjm,hip,him,up,um,vp,vm,
     $ zer,utt,uqt,umz,uno,due,tre,xxx,den,tichi,api,ame,u_p,u_m

      zer = 0.0d0
      utt = 0.125d0
      uqt = 0.25d0
      umz = 0.5d0
      uno = 1.0d0
      due = 2.0d0
      tre = 3.0d0
      ider = ider_w*ider_e*ider_n


      do k = 1,ider                    ! if(ider.eq.1) tutte centrate !
        do j = 1,my1
          do i = 2,mx1
            include'uce.f'
          end do
        end do
      end do
      do k = 1,(1-ider)              ! if(ider.eq.0) centrate in gruni !
        do j = 1,my1-jstre
          do i = istre_w+2,mx1-istre_e
           include'uce.f'
            include'usma.f'
          end do
        end do
      end do

      do k = 1,(1-ider_w)        ! if(ider_w.eq.0) up l zona stre west !
        do j = 1,my1-jstre
          do i = 2,istre_w+1
            include'uplu.f'
          end do
        end do
      end do
      do k = 1,ider_w        ! if(ider_w.eq.1) centrate zona stre west !
        do j = 1,my1-jstre
          i = 2
```

```
        include'uplu.f'
        do i = 2,istre_w+1
          include'uce.f'
        end do
      end do
    end do

    do k = 1,(1-ider_e)        ! if(ider_e.eq.0) up l zona stre est !
      do j = 1,my1-jstre
        do i = mx-istre_e,mx1
          include'uplu.f'
        end do
      end do
    end do
    do k = 1,ider_e        ! if(ider_e.eq.1) centrate zona stre est !
      do j = 1,my1-jstre
        do i = mx-istre_e,mx1
          include'uce.f'
        end do
        i = mx1
        include'uplu.f'
      end do
    end do

    do k = 1,(1-ider_n)        ! if(ider_n.eq.0) up l zona stre nord !
      do j = my-jstre,my1
        do i = 2,mx1
          include'uplu.f'
        end do
      end do
    end do
    do k = 1,ider_n        ! if(ider_n.eq.1) centrate zona stre nord !
      do j = my-jstre,my1
        do i = 2,mx1
          include'uce.f'
        end do
      end do
    end do


    return
    end
**************************************************************************
    subroutine vnoli(Hv,u,v,dx,dy,prof,proplo,smart,mx,mx1,mx2,my,
   $          my1,my2,istre_w,istre_e,jstre,ider_w,ider_e,ider_n)
    integer mx,mx1,mx2,my,my1,my2,istre_w,istre_e,jstre,ider,
   $ ider_w,ider_e,ider_n,i,j,k
```

```
      real*8 u(mx,0:my),v(0:mx,my),Hv(2,mx,my),prof(mx,my),smart,
     $ proplo(mx,my),dx(*),dy(*),hij,hjp,hjm,hip,him,up,um,vp,vm,
     $ zer,utt,uqt,umz,uno,due,tre,yyy,den,tichi,api,ame,v_p,v_m
c
      zer = 0.0d0
      utt = 0.125d0
      uqt = 0.25d0
      umz = 0.5d0
      uno = 1.0d0
      due = 2.0d0
      tre = 3.0d0
      ider = ider_w*ider_e*ider_n


      do k = 1,ider                    ! if(ider.eq.1) tutte centrate !
        do j = 2,my1
          do i = 1,mx1
            include'vce.f'
          end do
        end do
      end do
      do k = 1,(1-ider)          ! if(ider.eq.0) centrate in zona gruni !
        do j = 2,my1-jstre
          do i = istre_w+1,mx1-istre_e
           include'vce.f'
            include'vsma.f'
          end do
        end do
      end do

      do k = 1,(1-ider_w)        ! if(ider_w.eq.0) up l zona stre west !
        do j = 2,my1-jstre
          do i = 1,istre_w
            include'uplv.f'
          end do
        end do
      end do
      do k = 1,ider_w        ! if(ider_w.eq.1) centrate zona stre west !
        do j = 2,my1-jstre
          i = 1
          include'uplv.f'
          do i = 1,istre_w
            include'vce.f'
          end do
        end do
      end do
```

```
      do k = 1,(1-ider_e)       ! if(ider_e.eq.0)  up l zona stre est !
        do j = 2,my1-jstre
          do i = mx-istre_e,mx1
            include'uplv.f'
          end do
        end do
      end do
      do k = 1,ider_e         ! if(ider_e.eq.1) centrate zona stre est !
        do j = 2,my1-jstre
          do i = mx-istre_e,mx1
            include'vce.f'
          end do
          i = mx1
          include'uplv.f'
        end do
      end do

      do k = 1,(1-ider_n)       ! if(ider_n.eq.0) up l zona stre nord !
        do j = my-jstre,my1
          do i = 1,mx1
            include'uplv.f'
          end do
        end do
      end do
      do k = 1,ider_n       ! if(ider_n.eq.1) centrate zona stre nord !
        do j = my-jstre,my1
          do i = 1,mx1
            include'vce.f'
          end do
        end do
      end do


      return
      end
*************************************************************************
      subroutine vorti(vor,u,v,proplo,dx,dy,zer,uqt,umz,tre,ott,ove,
     $           mx,my,mx1,my1,mx2,my2,lip_w,lip_e,lip_s)
      integer mx,my,mx1,my1,mx2,my2,i,j,lip_w,lip_e,lip_s
      real*8 vor(mx,my),voplo(mx,my),proplo(mx,my),dx(*),dy(*),
     $       u(1:mx,0:my),v(0:mx,1:my),zer,uqt,umz,tre,ott,ove


      do j = 2,my1
        do i = 2,mx1
          voplo(i,j) = (v(i,j)-v(i-1,j))/(umz*(dx(i)+dx(i-1)))
     $               - (u(i,j)-u(i,j-1))/(umz*(dy(j)+dy(j-1)))
```

```
      end do
    end do

    do i = 2,mx1
      voplo(i,1) = (v(i,1)-v(i-1,1))/(umz*(dx(i)+dx(i-1)))
    $     - float(lip_s)*(-ott*u(i,0)+ove*u(i,1)-u(i,2))/(tre*dy(1))
      voplo(i,my) = (v(i,my)-v(i-1,my))/(umz*(dx(i)+dx(i-1)))
    $           - (ott*u(i,my)-ove*u(i,my1)+u(i,my2))/(tre*dy(my1))
    end do

    do j = 2,my1
      voplo(1,j) =
    $     float(lip_w)*(-ott*v(0,j)+ove*v(1,j)-v(2,j))/(tre*dx(1))
    $           - (u(1,j)-u(1,j-1))/(umz*(dy(j)+dy(j-1)))
      voplo(mx,j) =
    $  float(lip_e)*(ott*v(mx,j)-ove*v(mx1,j)+v(mx2,j))/(tre*dx(mx1))
    $           - (u(mx,j)-u(mx,j-1))/(umz*(dy(j)+dy(j-1)))
    end do

    voplo(1,1) = zer
    voplo(mx,1) = zer
    voplo(1,my) = zer
    voplo(mx,my) = zer

    do j = 1,my1
      do i = 1,mx1
        vor(i,j) = uqt*(voplo(i,j)/proplo(i,j)
    $               +voplo(i+1,j)/proplo(i+1,j)
    $               +voplo(i,j+1)/proplo(i,j+1)
    $               +voplo(i+1,j+1)/proplo(i+1,j+1))
      end do
    end do

    return
    end
**********************************************************************
    subroutine piuomeno(tichi,ame)
    real*8 zer,utt,ust,cqs,uno,due,tre,ame,tichi
    zer = 0.d0
    utt = 1.d0/8.d0
    ust = 1.d0/6.d0
    cqs = 5.d0/6.d0
    uno = 1.d0
    due = 2.d0
    tre = 3.d0
    if(tichi.lt.zer) then
      ame = (tichi-tre*(due*tichi+uno)*utt)/(due*tichi-uno)
```

```
      else if(tichi.ge.zer.and.tichi.lt.ust) then
        ame = (tre*tichi-tre*(due*tichi+uno)*utt)/(due*tichi-uno)
      else if(tichi.ge.ust.and.tichi.le.cqs) then
        ame = zer
      else if(tichi.gt.cqs.and.tichi.le.uno) then
        ame = (uno-tre*(due*tichi+uno)*utt)/(due*tichi-uno)
      else if(tichi.gt.uno) then
        ame = (tichi-tre*(due*tichi+uno)*utt)/(due*tichi-uno)
      end if
      return
      end
*************************************************************************
```