SAPIENZA
UNIVERSITÀ DI ROMA

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
MASTER DI II° LIVELLO IN CALCOLO SCIENTIFICO

Sviluppo di un solutore Shallow Water per lo studio del moto di vortici sul β-plane

Candidato:
Fabiana Mancini

Relatore interno:
Prof. Roberto Ferretti

Relatore esterno:
Dr Roberto Iacono

Anno Accademico 2014-2015
Dipartimento di Matematica 'Guido Castelnuovo'

Introduzione

Questa tesi presenta i principali risultati del lavoro di tirocinio svolto presso il Laboratorio Clima dell'ENEA (centro ricerche Casaccia). Il lavoro prende spunto dalle tematiche affrontate durante il corso del Master intitolato "Modelli numerici per il Clima", tra cui la rilevanza e l'uso di modelli semplificati per lo studio della dinamica atmosferica e/o oceanica.

Il modello su cui ci concentreremo è il modello Shallow Water, che è usato in un'infinità di problemi di geofluidodinamica e idraulica. Questo modello, in cui la densità del fluido è assunta omogenea e la dinamica è bidimensionale, fornisce nondimeno un contesto rilevante per l'analisi di molti problemi di fisica dell'atmosfera. In particolare, modelli di questo tipo sono stati usati e sono ancora usati per studiare le proprietà di vortici intensi, osservati sia in oceano che in atmosfera, che si propagano per lungo tempo, ritenendo la loro coerenza. In atmosfera, gli esempi più drammatici di tali vortici sono i cicloni. La previsione della loro intensità e della loro traiettoria rappresenta ancora oggi una sfida scientifica molto impegnativa, oltre ad avere una grande importanza socio-economica per i paesi esposti.

L'obiettivo di questo lavoro è stato dunque quello di sviluppare un modello per integrare nel tempo le equazioni Shallow Water e di utilizzarlo per effettuare simulazioni dell'evoluzione di vortici ciclonici con caratteristiche simili a quelle dei cicloni reali, per meglio comprendere la dipendenza della dinamica di questi vortici dalle condizioni iniziali.

La tesi è strutturata in tre capitoli. Nel primo sono riassunti alcuni concetti di base rilevanti, che hanno soprattutto a che fare con la vorticità e i modelli bidmensionali (equazione della vorticità barotropica, modello Shallow Water) che predicono la sua evoluzione. E' inoltre brevemente discusso

il problema dell'inizializzazione bilanciata e vengono ricordate alcune caratteristiche di base dei vortici ciclonici, per inquadrare l'obiettivo scientifico del lavoro. Nel secondo capitolo si descrivono i modelli numerici sviluppati, e cioè il codice per l'inizializzazione e il codice per l'integrazione nel tempo delle equazioni Shallow Water. Infine, nel terzo capitolo, si presentano i primi risultati delle simulazioni effettuate, che vengono confrontati con risultati di analoghe simulazioni descritte in letteratura. Il confronto mostra che il solutore Shallow Water sviluppato è perfettamente adeguato per affrontare il problema che ci siamo posto. Infine in appendice sono riportati i programmi sviluppati nel corso del lavoro di tesi.

Indice

Capitolo 1

Modelli bidimensionali per flussi geofisici

1.1 Vorticità e campo di velocità. Equazione della vorticità barotropica

Il più semplice modello rilevante per la descrizione della dinamica atmosferica è l'equazione della vorticità barotropica (in Inglese Barotropic Vorticity Equation, BVE nel seguito), che descrive l'evoluzione di un fluido omogeneo (ovvero con densità costante), non divergente (ovvero incomprimibile) sulla superficie di una sfera rotante. La BVE afferma semplicemente che la vorticità assoluta

$$\omega = \zeta + f$$

data dalla somma della vorticità relativa ζ più la vorticità planetaria f (anche detto parametro di Coriolis), è conservata lungo il flusso, ovvero che

$$\frac{D\omega}{Dt} = 0.$$

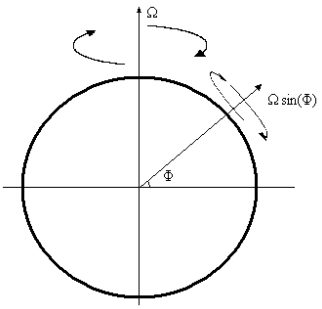
L'operatore D/Dt è l'operatore di derivata totale ed è dato da:

$$\frac{D}{Dt}(\ ) \equiv \frac{\partial}{\partial t}(\ ) + (\underline{v} \cdot \nabla)(\ )$$

dove $\underline{v}$ è la velocità del flusso (anche detto vento in ambito atmosferico). Il campo di velocità è espresso come $\underline{v} = (u, v)$, dove u è la componente in direzione longitudinale (zonale) mentre v è la componente in direzione meridionale. La vorticità relativa è legata al rotore del campo di velocità:

$$\zeta = \hat{k} \cdot \nabla \times \underline{v},$$

dove $\hat{k}$ è il versore normale alla superficie.
Vortici di tipo ciclonico, su cui ci focalizzeremo, sono caratterizzati da una vorticità relativa ζ positiva (almeno nella parte centrale del vortice). Nell'emisfero nord questo corrisponde ad una rotazione del vento in senso antiorario. Il parametro di Coriolis f è la parte di vorticità associata al moto di rotazione del pianeta. Se il pianeta ruota intorno al suo asse con velocità angolare Ω, ad una certa latitudine ϕ la superficie ruota con velocità angolare $\omega = \Omega \sin \phi$.



Quindi un fluido a riposo ha una vorticità perpendicolare alla superficie del pianeta data da

$$f = 2\omega = 2\Omega \sin \phi.$$

Osserviamo che f è zero all'equatore ed assume il valore massimo al polo. Per moti alle medie latitudini, ovvero tra i 30° e i 60° di latitudine nord si

prende spesso come riferimento $\phi = \frac{\pi}{4}$ e quindi essendo

$$\Omega = \frac{2\pi}{86400} s^{-1}$$

si ha

$$f = 2\omega = 2\Omega \sin \frac{\pi}{4} = 2\pi\sqrt{2}\frac{1}{86400} s^{-1} \sim 10^{-4},$$

mentre per la dinamica di vortici ciclonici che considereremo più avanti utilizzeremo valori più piccoli corrispondenti alla zona in cui vengono formati di solito i cicloni nell'Atlantico (10 - 15 gradi di latitudine nord).

1.1.1 f-plane e β-plane

La vorticità planetaria f è l'unico parametro geometrico che compare nelle equazioni fondamentali che governano i moti atmosferici ed oceanici. Per moti a scala globale, esso dipende dal seno della latitudine ϕ. Per moti che avvengono su porzioni della superficie terrestre si può proiettare in prima approssimazione il dominio di interesse sul piano tangente alla sfera e riscrivere le equazioni del moto su tale piano, usando coordinate cartesiane. In tal caso f deve essere opportunamente approssimato, e questo definisce le due principali approssimazioni utilizzate sia in ambito atmosferico che oceanico. Esse sono:

- l'approssimazione f-plane, valida su piccole scale (decine o centinaia di chilometri, moti di mesoscala), in cui il parametro di Coriolis viene assunto invariante rispetto alla latitudine. Viene posto cioè

  $$f = f_0 = 2\Omega \sin \phi_0$$

  dove ϕ_0 è la latitudine centrale della regione in considerazione. Si può ricorrere a tale approssimazione, per esempio, per studiare la dinamica in canali e stretti, che è l'oggetto di una specifica disciplina denominata "rotating hydraulics".

- l'approssimazione β-plane, valida per scale di moto di alcune migliaia di chilometri (le cosiddette scale sinottiche, che sono quelle che caratte-

rizzano per esempio le perturbazioni atmosferiche alle medie latitudini e lo stesso moto dei cicloni) in cui viene posto

$$f = f_0 + \beta y$$

dove

$$\beta = \frac{df}{dy} = 2\Omega \cos\phi_0 \frac{d\phi}{dy} = 2\Omega \cos\phi_0 \frac{1}{R}$$

dove R è il raggio della terra e $y = 0$ alla latitudine ϕ_0.

1.1.2 Una derivazione della BVE

Come precedentemente detto la BVE esprime la conservazione della vorticità assoluta:

$$\frac{\partial \zeta}{\partial t} + \underline{v} \cdot \nabla(\zeta + f) = 0. \tag{1.1}$$

Questa equazione può essere ricavata in vari modi, per esempio come limite non divergente di un modello bidimensionale più generale in cui l'equazione della quantità di moto è espressa come:

$$\frac{D\underline{v}}{Dt} + f\hat{k} \times \underline{v} = -\frac{1}{\rho}\nabla P$$

dove $P = gh\rho$ è la pressione, g l'accelerazione di gravità, h l'altezza locale del fluido e ρ la densità . Se assumiamo la densità costante, l'equazione precedente si riduce a

$$\frac{\partial \underline{v}}{\partial t} + \underline{v} \cdot \nabla \underline{v} + f\hat{k} \times \underline{v} = -g\nabla h \tag{1.2}$$

che è l'equazione della quantità di moto per il modello Shallow Water (SW), che discuteremo in seguito. Utilizzando una semplice identità vettoriale il termine avvettivo può essere riscritto come

$$\underline{v} \cdot \nabla \underline{v} = \frac{1}{2}\nabla \underline{v}^2 - \zeta \hat{k} \times \underline{v}$$

così che la (1.2) si riscrive

$$\frac{\partial \underline{v}}{\partial t} + (\zeta + f)\hat{k} \times \underline{v} = -\nabla(gh + \frac{1}{2}\underline{v}^2) = \\ = -\nabla B$$

dove $B = gh + \frac{1}{2}\underline{v}^2$ è detta la funzione di Bernoulli. Moltiplicando vettorialmente questa equazione per $\hat{k}$ e facendone poi la divergenza si ottiene

$$\frac{\partial \zeta}{\partial t} + \nabla \cdot [(\zeta + f)\underline{v}] = 0,$$

che è l'equazione di conservazione della vorticità nel modello SW. Nel limite di divergenza nulla essa si riduce alla BVE.

1.1.3 Caratterizzazione del campo di velocità

Consideriamo un flusso non divergente in geometria cartesiana. L'equazione

$$\nabla \cdot \underline{v} = 0$$

ha come soluzione

$$\underline{v} = \hat{k} \times \nabla \Psi$$

dove la funzione $\Psi = \Psi(x, y)$ è una generica funzione delle coordinate (x, y). Tale funzione è detta funzione di flusso o "stream function". Data questa funzione, il campo di velocità è univocamente determinato:

$$\begin{aligned} \underline{v} &= \hat{k} \times \nabla \Psi = \\ &= \hat{k} \times \left[\frac{\partial \Psi}{\partial x}\hat{i} + \frac{\partial \Psi}{\partial y}\hat{j} \right] = \\ &= \left[-\frac{\partial \Psi}{\partial y}\hat{j}, \frac{\partial \Psi}{\partial x}\hat{i} \right] = [u, v]\,. \end{aligned}$$

Ora se moltiplichiamo $\underline{v}$ per $\hat{k}$ otteniamo

$$\hat{k} \times \underline{v} = -\nabla \Psi$$

e facendone la divergenza

$$\nabla \cdot \left(\hat{k} \times \underline{v}\right) = -\nabla \cdot (\nabla \Psi) = -\Delta \Psi.$$

Ma osserviamo anche che

$$\nabla \cdot \left(\hat{k} \times \underline{v}\right) = \underline{v} \cdot \nabla \times \hat{k} - \hat{k} \cdot \nabla \times \underline{v} = -\zeta.$$

Dalle ultime due relazioni si ottiene l'equazione:

$$\Delta \Psi = \zeta$$

che collega vorticità e stream function. Questa equazione di Poisson deve essere risolta nel modello BVE per ricavare il campo di velocità da quello di vorticità che viene avanzato nel tempo dall'equazione (1.1). Quindi nella risoluzione numerica della BVE è necessario anche risolvere un problema ellittico (e quindi non-locale) per poter determinare il campo di velocità ad un dato istante.

1.2 Equazioni Shallow Water

Finora abbiamo trattato il caso di un fluido non divergente descritto dall'equazione di vorticità barotropica. Dall'ipotesi di divergenza nulla consegue che le equazioni per il campo di pressione e quello di velocità sono disaccoppiate; in questo caso il campo di pressione può essere calcolato a posteriori per motivi diagnostici utilizzando (1.2). Un modello più generale che ammette moti divergenti è il modello Shallow Water (SW), che per un flusso in assenza di topografia è dato da:

$$\begin{cases} \frac{\partial \underline{v}}{\partial t} + \underline{v} \cdot \nabla \underline{v} + f\hat{k} \times \underline{v} = -g\nabla h \\ \frac{\partial h}{\partial t} + \nabla \cdot (h\underline{v}) = 0. \end{cases} \tag{1.3}$$

Quindi, oltre all'equazione per la quantità di moto, che collega la variazione della velocità alle forze che agiscono sull'elemento di fluido (forza di Coriolis e

gradiente di pressione), abbiamo una seconda equazione, che è semplicemente l'equazione di continuità per h ($\rho h(x, y)$ è la massa della colonna di fluido nel punto (x, y)). Abbiamo quindi tre variabili prognostiche nel modello SW: lo spessore del fluido h e due variabili scalari che descrivono il campo di velocità. Tali variabili possono essere scelte in vari modi; per esempio, si può usare la coppia (ζ,δ), dove ζ è la vorticità relativa, già definita, e δ è la divergenza del campo di velocità. La scelta più comune, che adotteremo nel seguito, è quella della coppia di componenti (u,v). Con questa scelta, il sistema SW diventa:

$$\begin{cases} \frac{\partial u}{\partial t} + \underline{v} \cdot \nabla u = fv - g\frac{\partial h}{\partial x} \\ \frac{\partial v}{\partial t} + \underline{v} \cdot \nabla v = -fu - g\frac{\partial h}{\partial y} \\ \frac{\partial h}{\partial t} + \underline{v} \cdot \nabla h = -h\nabla \cdot \underline{v}. \end{cases}$$

1.2.1 Inizializzazione

Mentre per definire completamente le condizioni iniziali per la BVE basta la stream function iniziale o la vorticità iniziale, per le SW serve fornire oltre al campo di velocità iniziale la distribuzione iniziale dello spessore del fluido $h_0(x)$. Ciò è fonte di complicazione, poiché il sistema SW, a differenza della BVE, ammette differenti scale di moto. Oltre ai moti "lenti", che caratterizzano l'evoluzione del campo di vorticità, esistono anche moti molto più veloci (onde di gravità), che saranno in generale innescati da condizioni iniziali che non siano esse stesse soluzioni "lente" del sistema. Questo è un problema ben noto anche in ambito operativo previsionale, poiché le perturbazioni veloci innescate dall'inizializzazione si possono propagare in pochi giorni in tutto il dominio computazionale, deteriorando i campi di previsione. Sono state sviluppate varie tecniche per ovviare a questo problema. Una possibilità è quella di inizializzare il sistema in modo tale che per un certo tempo i moti legati alla parte non divergente del campo di velocità abbiano un'ampiezza piccola. A tal fine, si richiede che la tendenza della divergenza del flusso sia nulla all'istante iniziale. Ovvero applichiamo la divergenza all'equazione

della quantità di moto

$$\frac{\partial \nabla \cdot \underline{v}}{\partial t} = \nabla \cdot \left(-(\zeta + f)\hat{k} \times \underline{v} - \nabla(\frac{1}{2}\underline{v}^2) - g\nabla h \right),$$

e dobbiamo imporre che il termine a destra dell'equazione si annulli all'istante iniziale. Pertanto dato un campo di velocità iniziale ciò implica la risoluzione della seguente equazione di Poisson per h :

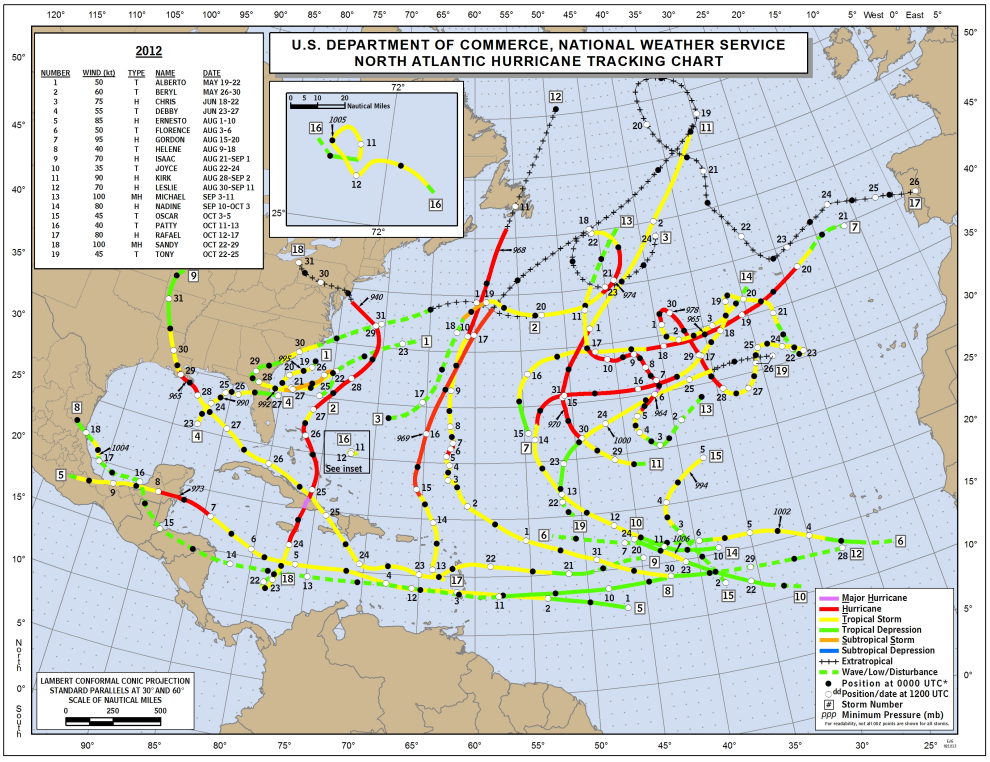
$$\begin{aligned} \Delta h &= \frac{1}{g}\nabla \cdot \left(-(\zeta + f)\hat{k} \times \underline{v} - \nabla(\frac{1}{2}\underline{v}^2) \right) = \\ &= \frac{1}{g}\left[\frac{\partial}{\partial x}\left((\zeta + f)v\right) - \frac{\partial}{\partial y}\left((\zeta + f)u\right) - \Delta\left(\frac{1}{2}\underline{v}^2\right) \right]. \end{aligned} \tag{1.4}$$

1.3 Moto dei cicloni

L'obiettivo principale del presente lavoro è lo sviluppo di un solutore numerico delle equazioni SW sul β-plane, che sarà descritto in dettaglio nel prossimo capitolo. E' però opportuno spiegare prima brevemente quali sono i problemi scientifici che hanno suggerito tale sviluppo. La motivazione di base è quella di avere uno strumento numerico capace di caratterizzare accuratamente (sia pure in ambito barotropico, e cioè non tenendo conto della dinamica verticale) il moto di vortici ciclonici che assomiglino ai cicloni che si osservano in atmosfera, e che producono molti danni in varie zone del pianeta. Ci interessano in particolare i cicloni atlantici, detti anche uragani, che nascono poco sopra l'Equatore, ad ovest del continente africano, e che possono attraversare l'oceano in un paio di settimane per poi abbattersi sui Caraibi o sul Nord America. Le traiettorie di questi cicloni, che da decenni vengono monitorate, sono molto varie, e sono determinate da diversi e complessi fenomeni fisici e termodinamici. In molti casi per capire queste traiettorie è necessario studiare gli scambi verticali di quantità di moto, di calore, e di umidità, che sono anche essenziali per capire l'intensificazione o il decadimento dei vortici. In molti altri, però, anche modelli bidimensionali, puramente fisici, possono fornire una buona descrizione della traiettoria. Tuttora modelli di questo tipo vengono utilizzati a scopo previsionale, e i loro

risultati vengono confrontati con quelli di modelli tridimensionali operativi (di complessità paragonabile, se non superiore, a quella dei modelli usati per le previsioni del tempo a scala globale), anche per capire il valore aggiunto di questi ultimi. Come si vede dalla Figura 1.1 (tratta dall'archivio del Natio-

Figura 1.1: Uragani della stagione 2012



nal Hurricane Center americano), che mostra tutti gli uragani della stagione 2012, alcune traiettorie sono "semplici", e mostrano, per molti giorni, una propagazione essenzialmente verso ovest, con una piccola componente in direzione nord. Altre volte, i cicloni partono rapidamente in direzione nord-ovest, che è quella considerata tipica per questo tipo di vortici, in un'atmosfera a riposo. In generale, però, il ciclone non si muove in un ambiente quiescente, ma attraversa sistemi di venti permanenti, anche molto forti, che caratterizzano la nostra atmosfera e che possono semplicemente "trasportare" con se il vortice. Questi venti, approssimativamente orientati in direzionale zona-

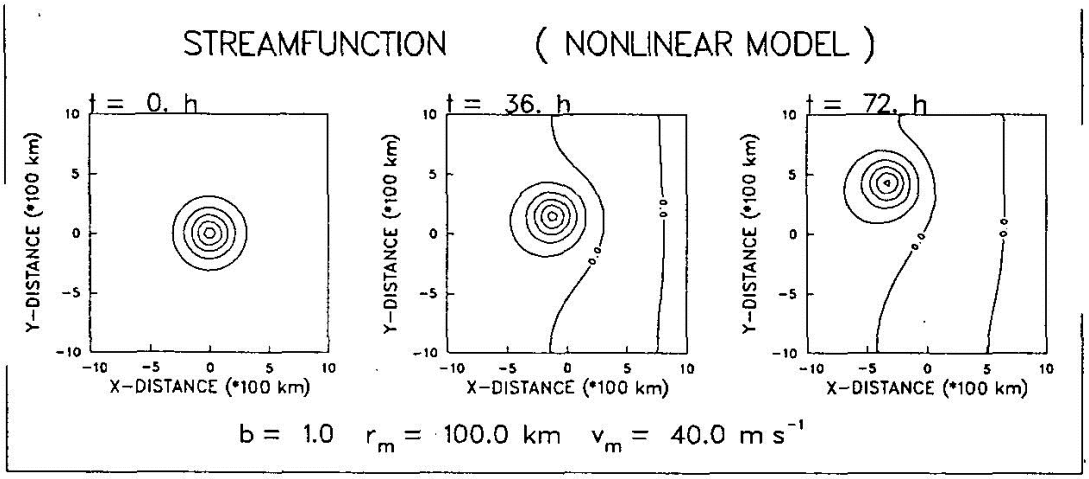
le, hanno anche forti gradienti meridionali, e anche questi gradienti possono influenzare il movimento del ciclone, anche se in modi più difficili da comprendere e analizzare. D'altra parte, è ben noto, anche se l'argomento non è stato studiato in modo approfondito, che anche le condizioni iniziali, e cioè la struttura iniziale del vortice quando comincia a propagarsi, possono influire sulla traiettoria.
Come si vede, il problema è molto complicato. Qui noi abbiamo cercato di affrontare un singolo aspetto, nella situazione più semplice possibile, e cioè quella di un vortice ciclonico bidimensionale che si muove in un'atmosfera quiescente. Consideriamo la propagazione di questo vortice in un β-plane centrato sulle latitudini a cui i cicloni atlantici sono generati, cercando di capire come i parametri iniziali del vortice influenzino la sua traiettoria iniziale. Un modello adeguato per affrontare questo problema è il modello Shallow Water, e questo ha motivato il lavoro numerico che descriveremo nel prossimo capitolo.

1.3.1 Un lavoro di riferimento

Come si può immaginare, negli ultimi decenni molti lavori, sia analitici che numerici, hanno cercato di caratterizzare il movimento di vortici ciclonici, in una varietà di situazioni, utilizzando modelli numerici via via più complicati. I primi lavori modellistici, nella seconda metà degli anni ottanta, e nella prima metà dei novanta, hanno utilizzato il modello dinamico più semplice, e cioè la BVE su un β-plane.
Va qui ricordato che la presenza di una variazione meridionale del parametro di Coriolis era stata riconosciuta da molto tempo, fin da lavori dello stesso Rossby (1939, 1948) ([1], [2]) come elemento indispensabile per la propagazione del ciclone. Questo punto è ben illustrato in un lavoro importante di Chan e Williams [3], basato sulla BVE, nel quale vengono effettuate simulazioni sia su un f-plane che su un β-plane. Nelle prime si vede che il vortice iniziale, di simmetria circolare, non si propaga, ma viene progressivamente elongato verso ovest, a causa dello sviluppo di onde di Rossby (che si propagano per l'appunto verso ovest), e perde rapidamente la sua coerenza. Invece,

in presenza di un β realistico, il vortice comincia dopo poco ad accelerare in direzione nord-ovest, mantenendo la sua coerenza, e raggiunge dopo un paio di giorni velocità di propagazione pressoché costanti (dipendenti dalla configurazione iniziale del vortice) di alcuni metri al secondo, che sono quelle tipiche dei cicloni osservati in natura. La Figura 1.2 mostra il profilo della stream function in quest'ultimo caso, ovvero in presenza del β. C'è propagazione e si vede anche che tale propagazione è associata allo sviluppo di una asimmetria.

Figura 1.2: Stream function nel caso in cui c'è β

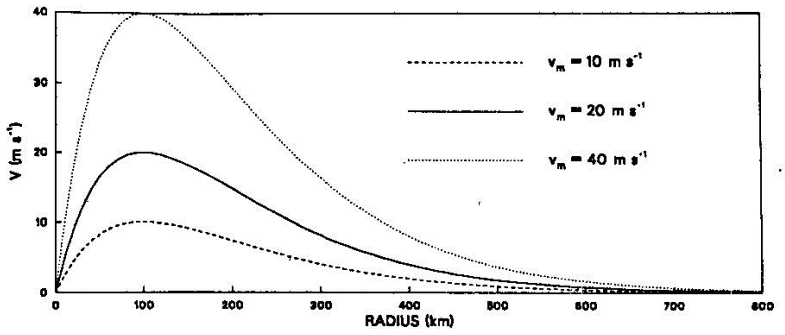


Chan e Williams cercano poi di capire come la velocità di propagazione (detta anche velocità di drift) e la direzione di propagazione dipendano dalle principali caratteristiche del vortice iniziale, come la sua dimensione, l'intensità del vento, e la forma del profilo radiale di velocità. In realtà, la dimensione dalla forma del profilo non viene studiata, poiché, anche se essi danno una forma del profilo iniziale di velocità dipendente da un parametro che può cambiare la forma del profilo stesso, nelle loro simulazioni fissano il valore del parametro, considerando un profilo della forma:

$$V(r) = V_m \left(\frac{r}{r_m}\right) exp\left\{1 - \left(\frac{r}{r_m}\right)\right\}, \tag{1.5}$$

dove il raggio r è la distanza dal centro del ciclone, V_m è il valore massimo della velocità tangenziale $V(r)$ e r_m è il raggio a cui essa viene raggiunta. Un esempio di questo profilo è mostrato nella Figura (1.3), che mostra il profilo di V(r) per un dato r_m uguale a 100 Km e tre differenti valori di V_m. Si può notare che i profili sono pressoché lineari vicino al centro del vortice, che ruota quindi con velocità angolare costante, e decadono abbastanza rapidamente al di fuori del raggio di massima velocità, caratteristica questa che è un po' discutibile, visto che i profili di vento tipici degli uragani tendono a decadere più lentamente. Dato il profilo (1.5), Chan e Williams analizzano la dipen-

Figura 1.3: Profilo del vento tangenziale



denza del moto del ciclone dai due parametri r_m e V_m, anche se in modo non molto dettagliato a causa probabilmente dei limiti del loro modello numerico e, soprattutto, della limitata potenza di calcolo a disposizione. Nell'ultimo capitolo faremo una simile analisi basata sugli output del solutore SW, e ci confronteremo con i loro risultati.

Capitolo 2

Codici numerici sviluppati

In questo capitolo presentiamo i codici numerici scritti in Fortran 90 che sono stati sviluppati nel corso del tirocinio. Il primo codice risolve il problema di inizializzazione, e cioè dato un vortice iniziale determina lo spessore del fluido necessario ad avere un'inizializzazione bilanciata in cui la tendenza iniziale della divergenza della velocità sia nulla. Questo codice fornisce le condizioni iniziali al secondo programma sviluppato, che integra nel tempo le equazioni SW sul β-plane.

2.1 Programma per l'inizializzazione

Il programma di inizializzazione, dato un campo di velocità in coordinate cartesiane (x, y), calcola lo spessore del fluido $h(x, y)$ necessario per una inizializzazione bilanciata, risolvendo l'equazione di Poisson (1.4) sul β-plane. Il programma lavora su un dominio quadrato centrato nell'origine del β-plane. Vari test sono stati effettuati con dimensioni del dominio da 2000×2000 Km a 8000×8000 Km, con passi di griglia $h_x = h_y$ da 10 a 40 Km.
Sono stati fissati i parametri f_0 e β, scegliendo come riferimento la latitudine di 10° nord . Abbiamo quindi che :

$$f = 2\Omega sin\frac{\pi}{18} = 0.253 \times 10^{-4}$$

$$\beta = \frac{1}{R}\frac{df}{d\phi} = \frac{2\Omega}{R} cos\frac{\pi}{18} = 2.253 \times 10^{-11}.$$

Data la condizione iniziale per la velocità, il programma calcola per prima cosa le derivate necessarie a valutare sul grigliato il termine di sorgente dell'equazione di Poisson (1.4), $S(x, y)$ dato da

$$S(x,y) = \frac{1}{g}\left\{\frac{\partial}{\partial x}\left[(\zeta + f)\, v - \frac{1}{2}\frac{\partial}{\partial x}V^2\right] + \frac{\partial}{\partial y}\left[-(\zeta + f)\, u - \frac{1}{2}\frac{\partial}{\partial y}V^2.\right]\right\}$$

Ricordiamo che, in geometria cartesiana,

$$\zeta(x,y) = \frac{\partial v}{\partial x}(x,y) - \frac{\partial u}{\partial y}(x,y).$$

Le derivate sono calcolate utilizzando formule alle differenze finite centrate al secondo ordine. Quindi dato un generico punto (i, j) del grigliato computazionale, le derivate parziali prime per una generica funzione $g(x, y)$ sono date da:

$$\frac{\partial g}{\partial x}(i,j) = \frac{g(i+1,j) - g(i-1,j)}{2h_x}$$

$$\frac{\partial g}{\partial y}(i,j) = \frac{g(i,j+1) - g(i,j-1)}{2h_y}.$$

Analogamente per le derivate seconde utilizzeremo le formule:

$$\frac{\partial^2 g}{\partial x^2}(i,j) = \frac{g(i+1,j) - 2g(i,j) + g(i-1,j)}{(h_x)^2}$$

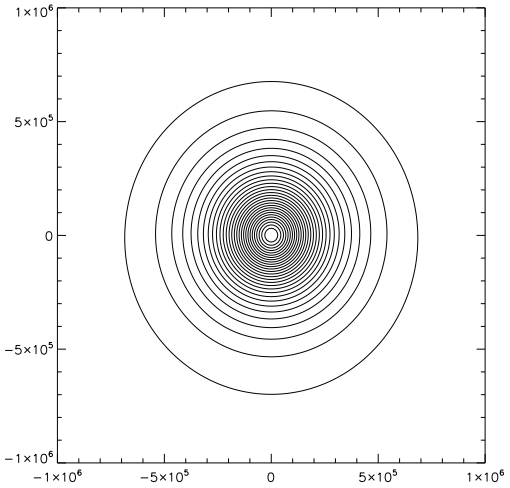
$$\frac{\partial^2 g}{\partial y^2}(i,j) = \frac{g(i,j+1) - 2g(i,j) + g(i,j-1)}{(h_y)^2}.$$

Ricordiamo infine che per utilizzare il programma di inizializzazione nel caso di un campo iniziale di velocità a simmetria radiale, come ad esempio quello del vortice ciclonico descritto da Chan e Williams (1.5), è stato necessario prima definire le componenti della velocità nel piano cartesiano utilizzando le formule di trasformazione:

$$\begin{cases} u = -V sin\theta = -Vy/\sqrt{x^2+y^2} \\ v = V cos\theta = Vx/\sqrt{x^2+y^2} \end{cases}$$

dove $V(r)$ è il profilo radiale di velocità del vortice. Dopo aver calcolato il termine di sorgente, il programma di inizializzazione chiama un solutore di libreria che può risolvere l'equazione di Helmholtz, e quindi anche il caso speciale dell'equazione di Poisson sia in geometria cartesiana che sulla sfera. Qualche dettaglio sulla struttura del solutore è dato in Appendice B. Visto che il solutore opera su un dominio quadrato $[0,1] \times [0,1]$, è stato necessario normalizzare opportunamente l'equazione di Poisson in input. Il campo dell'anomalia di h calcolato dal solutore, insieme al campo di velocità, vengono poi memorizzati su dei file che saranno poi utilizzati dal solutore SW. Un esempio di soluzione è mostrato in Figura 2.1:

Figura 2.1: Esempio di soluzione h del solutore di Poisson



2.2 Solutore per le equazioni Shallow Water

Questo programma integra nel tempo le equazioni Shallow Water su un β-plane a partire da una data condizione iniziale, che può essere o generata internamente dal codice a scopo di test, o letta dall'esterno (caso delle simulazioni della dinamica di vortici ciclonici).
Il dominio computazionale è un dominio quadrato di lato L, con centro nel-

l'origine del β-plane. Diverse scelte delle condizioni al bordo sono possibili. In particolare per le simulazioni del moto dei cicloni è possibile utilizzare :

- condizioni periodiche in direzione zonale e dominio chiuso al bordo superiore e inferiore, in modo da simulare il moto in un canale periodico
- condizioni periodiche in entrambe le direzioni
- bacino chiuso con "sponge layers" vicino ai bordi in cui si impone una forte diffusività numerica per smorzare le onde che si propagano nel dominio
- condizioni non periodiche più complesse (condizioni radiative) in cui i bordi sono aperti.

Tutte queste condizioni sono state utilizzate in letteratura per lo studio del moto dei cicloni. Per il momento abbiamo scelto di implementare le condizioni più semplici che sono quelle periodiche in entrambe le direzioni. In ogni caso per le simulazioni del moto di cicloni scegliamo le dimensioni del dominio sufficientemente grandi ($L \geq 4000$ Km) in modo da avere il vortice sufficientemente lontano dai bordi del dominio per diversi giorni.
Diverse scelte sono anche possibili per gli schemi numerici per la discretizzazione degli operatori spaziali e per l'avanzamento nel tempo delle equazioni. Anche qui abbiamo scelto una soluzione semplice e cioè quella di utilizzare schemi espliciti di alto ordine sia per la discretizzazione spaziale che per quella temporale. In particolare usiamo differenze finite centrate del quarto ordine nello spazio e uno schema Runge-Kutta del quarto ordine per le derivate temporali.
Poiché questi schemi sono poco diffusivi, è necessario introdurre una diffusività numerica esplicita per poter tenere sotto controllo la crescita del rumore prodotto sulla scala del passo di griglia dagli errori di discretizzazione e amplificati dai termini non lineari. Consistentemente con gli schemi numerici adottati abbiamo quindi introdotto termini di diffusione bi-armonici in ognuna delle tre equazioni con coefficienti scelti in base a sperimentazione.
Nel seguito descriviamo brevemente la struttura generale del codice.

2.2.1 Struttura del solutore SW

Il programma ha un breve "main" che chiama le seguenti routine:

- setio: apre file di input da cui vengono lette le condizioni iniziali necessarie per il run, e di output.
- conini: legge i dati che definiscono la simulazione: *ncase*, che definisce il tipo di simulazione (ncase=1 per un run di test, ncase=2 per un run che legge condizioni iniziali dall'esterno); *dt*, il time step usato per la simulazione; *nsteps*, il numero di time steps da effettuare; *dissip*, coefficiente dei termini di iperdiffusione; LX,LY, dimensioni del dominio computazionale; f_0, parametro di Coriolis a $y = 0$; β, parametro β. La routine procede poi a calcolare parametri derivati come i passi del grigliato nella direzione x e y, i coefficienti numerici per le formule delle derivate spaziali e temporali, e il parametro f su tutto il dominio.
- initc : se ncase=1, questa subroutine genera una soluzione analitica stazionaria delle equazioni SW che viene usata come test dell'accuratezza dell'integrazione numerica (vedi appendice A). Se ncase=2 la subroutine legge da file i valori iniziali di u,v,h.

Ciò fatto si procede all'integrazione con un loop temporale che chiama al suo interno una routine (steprk) che avanza nel tempo le equazioni utilizzando uno schema Runge-Kutta al quarto ordine e effettua gli output necessari. La routine steprk chiama a sua volta altre routine (rhs, deriva), che calcolano gli operatori spaziali necessari.

Capitolo 3

Simulazioni numeriche del moto di cicloni

Come primo test abbiamo cercato di replicare i risultati del lavoro di Chan e Williams [3]. A tal fine, sono stai effettuati vari test per determinare i valori ottimali della dimensione del dominio computazionale e del passo di griglia. Si è visto che nel range di parametri esplorato, l'evoluzione dei vortici è essenzialmente invariata quando il lato del dominio passa da 4000 a 8000 Km. I risultati che vengono mostrati sono stati quindi ottenuti con un dominio di 4000×4000 Km. Riguardo al passo del grigliato, vi sono differenze passando da 40 Km a 20 Km, e differenze molto più piccole passando da 20 Km a 10 Km. Tuttavia al fine di determinare con accuratezza la posizione del centro del vortice a un dato istante di tempo senza dover ricorrere a interpolazioni bidimensionali di alto ordine, le simulazioni sono state effettuate con un passo di griglia di 10 Km. La Figura 3.1 mostra la traiettoria di un ciclone con $V_m = 40$ m/s e $r_m = 100$ Km (profilo di velocità (1.5)). A destra è il risultato della simulazione SW mentre a sinistra è la traiettoria dal lavoro di Chan e Williams. In entrambi i grafici i simboli designano la posizione del centro del vortice ogni 12 ore. Come si può vedere c'è ottimo accordo tra le traiettorie nelle due simulazioni. La figura mostra che dopo una fase iniziale di accelerazione il vortice comincia a propagarsi con una velocità pressoché costante. Simulazioni con lo stesso valore di V_m ma differenti valori di r_m

mostrano che la direzione della traiettoria cambia molto poco in accordo con i risultati di Chan e Williams (vedi Tabella 1, [3]).
Abbiamo poi effettuato numerose simulazioni per determinare la dipendenza della velocità di drift dai parametri V_m e r_m. I risultati sono riassunti nella Figura 3.2, che come la precedente mostra a destra i risultati del modello SW e a sinistra quelli di Chan e Williams. Anche se c'è accordo tra i due grafici, i nostri risultati mostrano degli andamenti più dettagliati dovuti al maggior numero di run effettuati e alla risoluzione spaziale più fine. Nella figura i risultati numerici SW sono confrontati con i valori dedotti da una legge di scala proposta da Smith (1993 [7]) che sono rappresentati dalle curve continue. Secondo questo scaling, basato su considerazioni dimensionali e sulle simulazioni di Chan e Williams, la velocità di drift è data approssimativamente da

$$V_d = 0.9\, r_m \sqrt{V_m \beta}. \tag{3.1}$$

Il grafico mostra che, nonostante alcune differenze, la legge di scala di Smith è in buon accordo con i risultati numerici.

Figura 3.1: traiettoria del centro del vortice

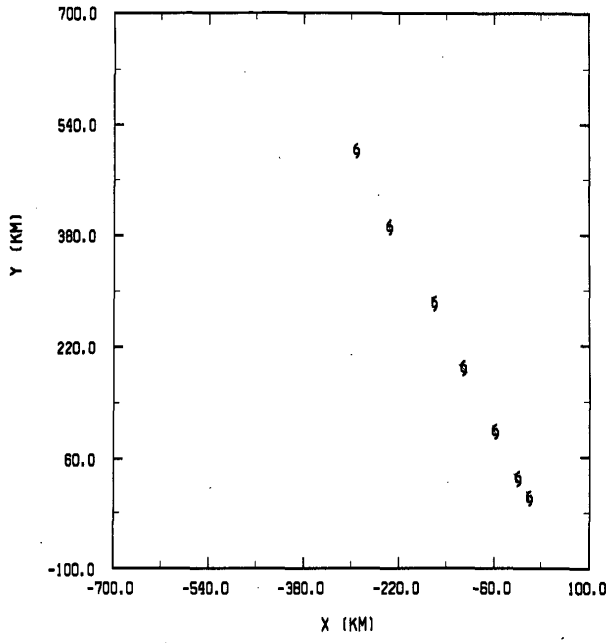
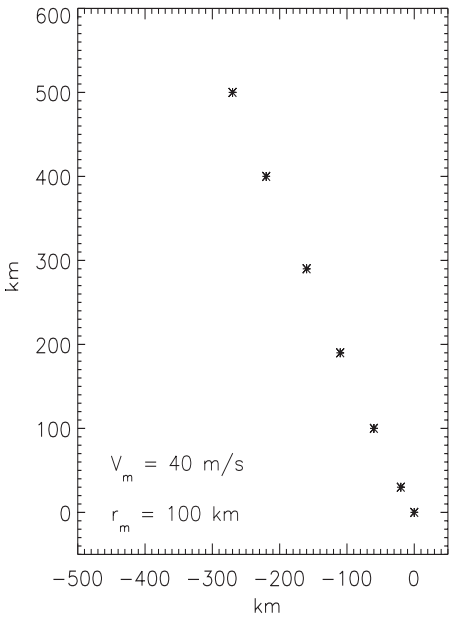


FIG. 6. The 0–72 h track of the vortex with r_m = 100 km, V_m = 40 m s^{-1}. The symbols along the track are 12 h apart.



Stessa figura dal modello SW

Figura 3.2: Velocità di drift del vortice in funzione di V_m ed r_m

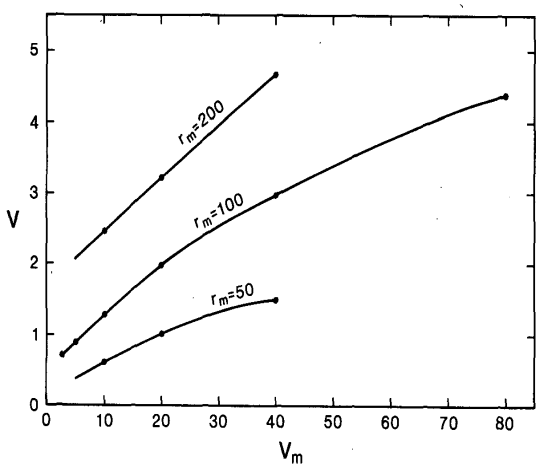
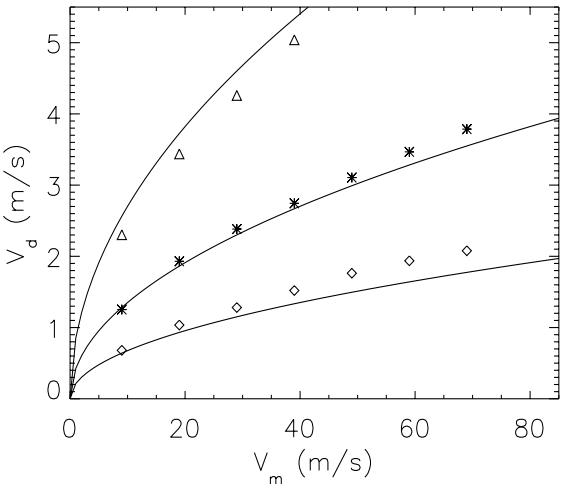


FIG. 8. The speed of movement V (m s^{-1}) as a function of V_m (m s^{-1}) for three values of r_m (km).



Stessa figura dal modello SW

Le curve continue rappresentano lo scaling di Smith (93)

Appendice A

Test case

Come test case per il solutore SW utilizziamo una soluzione analitica stazionaria valida per $f = cost$. In particolare scegliamo una soluzione che rappresenta un flusso zonale che varia in direziona meridionale, $u = u(y)$. Il valore di $h(y)$ corrispondente viene calcolato integrando l'equazione:

$$g\frac{\partial h}{\partial y}(y) = -fu(y).$$

Ottenendo quindi

$$gh(y) = cost - f\int_{-L}^{y} u(z)dz$$

dove L è la coordinata y del bordo inferiore del dominio.
Poiché abbiamo scelto condizioni al bordo periodiche utilizziamo un profilo $u(y)$ dato da:

$$u(y) = u_0 cos\left(\frac{\pi y}{L}\right).$$

Il valore di h corrispondente è dato da:

$$h(y) = h(-L) - \frac{fu_0 L}{g\pi} sin\left(\frac{\pi y}{L}\right).$$

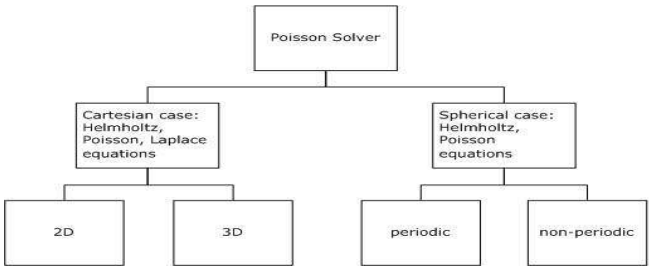
Appendice B

Poisson Solver

Poisson Solver Implementation

Poisson Solver routines enable approximate solving of certain two-dimensional and three-dimensional problems. Figure "Structure of the Poisson Solver" shows the general structure of the Poisson Solver.

Structure of the Poisson Solver



NOTE
Although in the Cartesian case, both periodic and non-periodic solvers are also supported, they use the same interfaces.

Sections below provide details of the problems that can be solved using Intel MKL Poisson Solver.

Two-Dimensional Problems

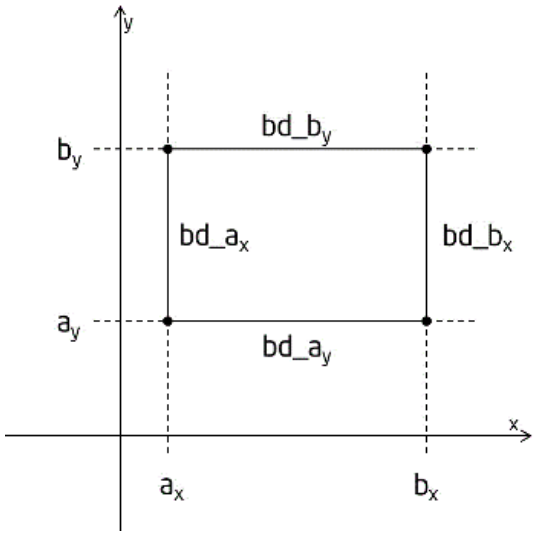
Notational Conventions

The Poisson Solver interface description uses the following notation for boundaries of a rectangular domain $a_x < x < b_x$, $a_y < y < b_y$ on a Cartesian plane:

$bd\_a_x = \{x = a_x, a_y \le y \le b_y\}$, $bd\_b_x = \{x = b_x, a_y \le y \le b_y\}$

$bd\_a_y = \{a_x \le x \le b_x, y = a_y\}$, $bd\_b_y = \{a_x \le x \le b_x, y = b_y\}$.

The following figure shows these boundaries:



The wildcard "+" may stand for any of the symbols a_x, b_x, a_y, b_y, so bd_+ denotes any of the above boundaries.

The Poisson Solver interface description uses the following notation for boundaries of a rectangular domain $a_\varphi < \varphi < b_\varphi$, $a_\theta < \theta < b_\theta$ on a sphere $0 \le \varphi \le 2\pi$, $0 \le \theta \le \pi$:

$bd\_a_\varphi = \{\varphi = a_\varphi, a_\theta \le \theta \le b_\theta\}$, $bd\_b_\varphi = \{\varphi = b_\varphi, a_\theta \le \theta \le b_\theta\}$,

$bd\_a_\theta = \{a_\varphi \le \varphi \le b_\varphi, \theta = a_\theta\}$, $bd\_b_\theta = \{a_\varphi \le \varphi \le b_\varphi, \theta = b_\theta\}$.

The wildcard "~" may stand for any of the symbols a_φ, b_φ, a_θ, b_θ, so $bd\_\sim$ denotes any of the above boundaries.

Two-dimensional Helmholtz problem on a Cartesian plane

The two-dimensional (2D) Helmholtz problem is to find an approximate solution of the Helmholtz equation

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} + qu = f(x,y),\ q = const \ge 0$$

in a rectangle, that is, a rectangular domain $a_x < x < b_x$, $a_y < y < b_y$, with one of the following boundary conditions on each boundary bd_+:

- The Dirichlet boundary condition
  $$u(x,y) = G(x,y)$$
- The Neumann boundary condition
  $$\frac{\partial u}{\partial n}(x,y) = g(x,y)$$
  where

  $n = -x$ on bd_a_x, $n = x$ on bd_b_x,

  $n = -y$ on bd_a_y, $n = y$ on bd_b_y.

- Periodic boundary conditions

$$u(a_x,y) = u(b_x,y),\ \frac{\partial}{\partial x}u(a_x,y) = \frac{\partial}{\partial x}u(b_x,y),$$

$$u(x,a_y) = u(x,\ b_y),\ \frac{\partial}{\partial y}u(x,a_y) = \frac{\partial}{\partial y}u(x,b_y).$$

Two-dimensional Poisson problem on a Cartesian plane

The Poisson problem is a special case of the Helmholtz problem, when q=0. The 2D Poisson problem is to find an approximate solution of the Poisson equation

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x,y)$$

in a rectangle $a_x < x < b_x$, $a_y < y < b_y$ with the Dirichlet, Neumann, or periodic boundary conditions on each boundary `bd_+`. In case of a problem with the Neumann boundary condition on the entire boundary, you can find the solution of the problem only up to a constant. In this case, the Poisson Solver will compute the solution that provides the minimal Euclidean norm of a residual.

Appendice C

inizializzazione.f90

```
!*****************************************************************
!-compute source term for initialization, for a given initial vortex
!-solve Poisson equation for delta(h)
!
! Reference calulation: Chan_Williams profile:
! V(r)=  Vm /rm * r * exp{ ( 1 - (r/rm) )}
!*****************************************************************

program inizializzazione

  implicit none

!********************  DEFINIZIONE PARAMETRI  *******************

! Definisco il numero di intervalli in cui voglio suddividere
! il segmento sull'asse x e y rispettivamente
! RICORDA DEVONO ESSERE VALORI PARI
  integer nx,ny,mx,my
!   parameter(nx=801, ny=801)
!   parameter(mx=800, my=800)
  parameter(nx=401, ny=401)
```

```
  parameter(mx=400, my=400)
!  parameter(nx=201, ny=201)
!  parameter(mx=200, my=200)
! ax, bx estremi inferiore e superiore  del dominio sull'asse x;
! ay, by estremi inferiore e superiore del dominio sull'asse y;
! lx,ly lunghezza dell'intervallo sull'asse x e asse y
! rispettivamente;
! hx, hy passo di discretizzazione;
! xi, yi punti del grigliato sull'asse x e y rispettivamente
! c distanza dal centro
  real ax, bx, ay, by, lx, ly, hx, hy, c

!Definisco dei parametri che serviranno per la velocita' di C-W
  real Vm , rm, x0, y0
  real beta, f_0, xi, yi, appo1

!Definiamo la costante di gravita'
  real, parameter :: g=9.822

! Definiamo indici per cicli do
  integer ix,iy,i, nx0, ny0

  integer :: ios

! Definiamo le matrici: modulo della velocita' V,
! componenti u_0 e v_0 del campo di velocita',
! matrice dei parametri di Coriolis f per il METODO 1
  real V(nx,ny), v0(nx,ny), u0(nx,ny), f(nx,ny)

! Definiamo le matrici A, B, tx, ty sx, sy per il METODO 2
  real A(nx,ny), B(nx,ny), tx(nx,ny), ty(nx,ny), sx(nx,ny), sy(nx,ny)

! Defininiamo i vettori derivate sui punti interni al
```

```
! rettangolo v0_x, u0_y, la vorticia' z,
! la derivata della vorticita'' rispetto alle x e y
  real v0_x(nx,ny), u0_y(nx,ny), z(nx,ny), z_x(nx,ny),
z_y(nx,ny)

! Definiamo il modulo della velocita' al quadrato e
! le sue derivate seconde
  real VV(nx,ny), VV_xx(nx-2,ny-2), VV_yy(nx-2,ny-2)

! Definiamo S, funzione a destra della nostra equazione
! di Laplace and deltah
  real S(nx,ny),deltah(nx,ny) , laplaciano_deltah(nx,ny)
  real verifica(nx,ny), deltah_xx(nx,ny), deltah_yy(nx,ny)

!************************************************************

! Definiamo il dominio rettangolare (ax,bx) x (ay, by)
!  ax=-4.E6
!  bx=4.E6
!  ay=-4.E6
!  by=4.E6

  ax=-2.E6
  bx=2.E6
  ay=-2.E6
  by=2.E6

! calcoliamo il passo di discretizzazione lungo la direzione x
  lx=bx-ax
  hx=lx/float(nx-1)
! calcoliamo il passo di discretizzazione lungo la direzione y
  ly=by-ay
  hy=ly/float(ny-1)
```

```
!******************** INIZIALIZZAZIONE VARIABILI E MATRICI
********

! define initial position of the vortex
  Vm=40.    !m/s
  rm=2.E5   !m

  nx0=(nx+1)/2
  ny0=(ny+1)/2

  V=0.
  VV=0.
  u0=0.
  v0=0.
  !u0_y=0.
  !v0_x=0.
  A=0.
  B=0.
  z=0.
  tx=0.
  ty=0.
  S=0.
  laplaciano_deltah(nx,ny)=0.

! f0 e beta a 10 gradi N, come in CW
  f_0=0.253E-4
  beta=2.254E-11

! Define initial vortex**************************************

  x0=0.
  y0=0.
```

```
! Calcoliamo i valori di V , u_0, v_0 sui punti della griglia
  do iy=1,ny
     do ix=1,nx
        xi=(ax+hx*(ix-1))
        yi=(ay+hy*(iy-1))

        if (xi.ne.x0.or.yi.ne.y0) then
           c=(xi-x0)**2 + (yi-y0)**2
           c=sqrt(c)
           V(ix,iy)= (Vm/rm) * c * exp(1-(c/rm))

           u0(ix,iy) = -V(ix,iy) * (yi-y0)/c
           v0(ix,iy) =  V(ix,iy) * (xi-x0)/c
           f(ix,iy)=f_0+beta*yi
           VV(ix,iy)=u0(ix,iy)**2+v0(ix,iy)**2
         endif
     enddo
  enddo

!*************************************************************

! Calcoliamo la derivata di v_0 rispetto le x
  do iy=2,ny-1
     do ix=2,nx-1

        v0_x(ix,iy)=(v0(ix+1,iy)-v0(ix-1,iy))/(2.*hx)

     enddo
  enddo

! Calcoliamo la derivata di u_0 rispetto le y
  do iy=2,ny-1
     do ix=2,nx-1
```

```
        u0_y(ix,iy)=(u0(ix,iy+1)-u0(ix,iy-1))/(2.*hy)

      enddo
    enddo

! Compute relative vorticity
    do iy=2,ny-1
      do ix=2,nx-1

        z(ix,iy)=v0_x(ix,iy)-u0_y(ix,iy)
      enddo
    enddo

!*************************************************************
! compute source term for the balance equation

! Calcoliamo la derivata di  V^2 rispetto le x
! e la matrice (z+f)*v0 - 1/2 * d/dx(V^2)
    do iy=2,ny-1
      do ix=2,nx-1

        A(ix,iy)=(VV(ix+1,iy)-VV(ix-1,iy))/(2.*hx)
        tx(ix,iy)=(z(ix,iy)+f(ix,iy))*v0(ix,iy)-0.5*A(ix,iy)

      enddo
    enddo

! Calcoliamo la derivata di V^2 rispetto le y
! e la matrice -(z+f)*u0 - 1/2 * d/dy (V^2)
    do iy=2,ny-1
      do ix=2,nx-1
```

```
        B(ix,iy)=(VV(ix,iy+1)-VV(ix,iy-1))/(2*hy)
        ty(ix,iy)=-(z(ix,iy)+f(ix,iy))*u0(ix,iy)-0.5*B(ix,iy)

      enddo
    enddo

! Calcoliamo la derivata di tx rispetto alle x
    do iy=2,ny-1
      do ix=2,nx-1

        sx(ix,iy)=(tx(ix+1,iy)-tx(ix-1,iy))/(2.*hx)

      enddo
    enddo


! Calcoliamo la derivata di ty rispetto alle y
    do iy=2,ny-1
      do ix=2,nx-1

        sy(ix,iy)=(ty(ix,iy+1)-ty(ix,iy-1))/(2.*hy)

      enddo
    enddo

!**************  CALCOLO DI S ************************************

    do iy=1,ny
      do ix=1,nx

        S(ix,iy)=(1/g)*( sx(ix,iy) + sy(ix,iy) )

      enddo
```

```
  enddo

!********** MEMORIZZAZIONE RISULTATI ********************************


open(10,FILE='source.dat',STATUS='REPLACE',ACTION='write',IOSTAT=ios)
  write(10,*) S
  close(10)

  print*,ios

  ! Call Poisson solver to solve:  Nabla (delta h) = S(x,y)

  call psolv(mx,my,S,deltah)

  open(unit=18, file='u.dat',action="write",status="replace")
  write(18,*) u0
  close(18)

  open(unit=19, file='v.dat',action="write",status="replace")
  write(19,*) v0
  close(19)

  open(unit=20, file='deltah.dat',action="write",status="replace")
  write(20,*) deltah
  close(20)


  !*************************************************************

  !$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

  !************* verifica ***************************************
```

```
  do iy=2,ny-1
     do ix=2,nx-1

        deltah_xx(ix,iy)=(deltah(ix+1,iy)- &
              2.*deltah(ix,iy)+deltah(ix-1,iy))/(hx)**2

     enddo
  enddo

  do iy=2,ny-1
     do ix=2,nx-1

        deltah_yy(ix,iy)=(deltah(ix,iy+1)- &
              2.*deltah(ix,iy)+deltah(ix,iy-1))/(hy)**2
        laplaciano_deltah(ix,iy)=deltah_xx(ix,iy)+deltah_yy(ix,iy)
     enddo
  enddo

  do iy=1,ny
     do ix=1,nx

        verifica(ix,iy)=(1/g)*( sx(ix,iy) + sy(ix,iy) ) &
             - laplaciano_deltah(ix,iy)

     enddo
  enddo

open(30, FILE='prova.dat',STATUS='REPLACE',ACTION='write',IOSTAT=ios)
  write(30,*) verifica
  close(30)
```

```
  stop
end program inizializzazione
!***************************************************************
! Subroutine PSOLV
!***************************************************************
subroutine psolv(mx,my,S,deltah)
  !*************************************************************
  !  Content:
  !  Fortran-90 single precision example of solving 2D Poisson problem
  !  in a rectangular domain using MKL Poisson Library
  !
  !*************************************************************
  !*************************************************************
  !
  ! Vogliamo risolvere l'equazione -Laplaciano ( H ) = S1
  ! La funzione a destra S1 e' una matrice calcolata in
INIT
  !
  !*************************************************************


  !include 'mkl_dfti.f90'
  !include 'mkl_poisson.f90'

  ! Include modules defined by mkl_poisson.f90 and
```

```
! mkl_dfti.f90 header files
use mkl_poisson
!use mkl_dfti

implicit none


!******** DEFINIZIONE DELLE VARIABILI E FUNZIONI ***************

! Definiamo nx e ny, NUMERO DI INTERVALLI  in cui suddividiamo
!  asse x e asse y rispettivamente
integer mx,my

! Definiamo indici per cicli do
integer ix, iy, i, stat


!Definiamo il vettore ipar che servira' nella chiamata di Helmholtz
integer ipar(128)


! ax, bx estremi inferiore e superiore  del dominio sull'asse x;
! ay, by estremi inferiore e superiore del dominio sull'asse y;
! lx,ly lunghezza dell'intervallo sull'asse x e asse y
! rispettivamente;
! hx, hy passo di discretizzazione;
! xi, yi punti del grigliato sull'asse x e y rispettivamente
real ax, bx, ay, by, lx, ly, hx, hy, xi, yi, c, c1, L


! Definiamo il vettore spar che servira' nelle chiamate Helmholtz
real spar(13*mx/2+7)
```

```
!Definiamo S matrice della funzione a destra
real S(mx+1,my+1)

real deltah(mx+1,my+1 )

! Definiamo le matrici
real LaplacianoH(mx+1,my+1 ), H_xx(mx+1,my+1 )
real H_yy(mx+1,my+1 ), S11(mx+1,my+1)


! Definiamo le variabili che assumeranno i valori delle condizioni
! al bordo
real bd_ax(my+1), bd_bx(my+1 ), bd_ay(mx+1 ), bd_by(mx+1)

! Definiamo il reale q il quale e' uguale a zero nell'equazione
! di Poisson, ma diverso da zero nell'equazione di Helmholtz
real q

!Definiamo un puntatore richiesto nelle chiamate di Helmholtz
type(DFTI_DESCRIPTOR), pointer :: xhandle

! Definiamo la stringa di quattro caratteri in cui si specifichera'
! che tipo di condizioni al bordo saranno adottate
! D: condizioni di Dirichlet; N: condizioni di Neumann;
! P: condizioni periodiche
character(4) BCtype

!*****************************************************************
!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

open(10, FILE='source.dat')
read(10,*) S
```

```
close(10)

ax=0.
bx=1.
ay=0.
by=1.

L=4.E6**2
!   L=8.E6**2

S=L*S
S11=S
!Diamo il valore al parametro a e psi_zero
!a=200.
!psi_zero=1.


!*****************************************************************
! Setting the coefficient q to 0.
! Note that this is the way to use Helmholtz Solver to
! solve Poisson problem!
!*****************************************************************
q=0.

! Computing the mesh size hx in x-direction
lx=bx-ax
hx=lx/float(mx)
! Computing the mesh size hy in y-direction
ly=by-ay
hy=ly/float(my)

BCtype = 'DDDD'
```

```
  ! Setting the values of the boundary function G(x,y)
  ! that is equal to the TRUE solution in the mesh points
  ! laying on Dirichlet boundaries
  do iy = 1,my+1
     bd_ax(iy) = 0.0
     bd_bx(iy) = 0.0
  enddo
  ! Setting the values of the boundary function g(x,y)
  ! that is equal to the normal derivative
  ! of the TRUE solution in the mesh points laying
  ! on Neumann boundaries
  do ix = 1,mx+1
     bd_ay(ix) = 0.0
     bd_by(ix) = 0.0
  enddo

  ! Initializing ipar array to make it free from garbage
  do i=1,128
     ipar(i)=0
  enddo

  ! Initializing simple data structures of Poisson Library
  ! for 2D Poisson Solver
call s_init_Helmholtz_2D(ax,bx,ay,by,mx,my,BCtype,q,ipar,spar,stat)
  if (stat.ne.0) goto 999

  ! Initializing complex data structures of Poisson Library
  ! for 2D Poisson Solver
  ! NOTE: Right-hand side f may be altered after the Commit step.
  ! If you want to keep it,you should save it in
  ! another memory location!
call s_commit_Helmholtz_2D(S,bd_ax,bd_bx,bd_ay,bd_by,&
                           xhandle,ipar,spar,stat)
```

```
  if (stat.ne.0) goto 999

  ! Computing the approximate solution of 2D Poisson problem
  ! NOTE: Boundary data stored in the arrays bd_ax, bd_bx, bd_ay,
  ! bd_by should not be changed
  ! between the Commit step and the subsequent
  ! call to the Solver routine!
  ! Otherwise the results may be wrong.
call s_Helmholtz_2D(S,bd_ax,bd_bx,bd_ay,bd_by,xhandle,ipar,spar,stat)
   if (stat.ne.0) goto 999

   ! Cleaning the memory used by xhandle
   call free_Helmholtz_2D(xhandle, ipar, stat)
   if (stat.ne.0) goto 999
   ! Now we can use xhandle to solve another 2D Poisson problem

   !****************************************
   ! change sign of deltah

   S=-S
   deltah=S
   ! Printing the results
   write(*,10) mx
   write(*,11) my
   print *, ''
   ! Watching the error along the line x=hx
   !!ix=nx/2+1
   !c1 = 0.0
   !do iy=1,ny+1
   !   write(*,12) (ix-1)*hx/lx, (iy-1)*hy/lx, z(ix,iy)-psi(ix,iy)
   ! if (abs(z(ix,iy)-psi(ix,iy)).ge.c1) c1 = abs(z(ix,iy)-psi(ix,iy))
   !enddo
   !print *, ''
```

```
   if (c1.ge.0.1E+0) then
      print *, 'The computed solution seems to be inaccurate.'
      goto 999
   endif

   ! Success message to print if everything is OK
   print *, ' real 2D Poisson example has successfully PASSED'
   print *, ' through all steps of computation!'
   ! Jumping over failure message
   go to 1
   ! Failure message to print if something went wrong
999 print *, 'real 2D Poisson example FAILED to compute the solution..

1  continue

10 format(1x,'The number of mesh intervals in x-direction is nx=',I4)
11 format(1x,'The number of mesh intervals in y-direction is ny=',I4)
12 format(1x,'In the mesh point (',F5.2,',',F5.2,') the error &
 between the computed and the true solution is equal to ', E10.3)

! End of the example code
   !Inizializzo a zero le seguenti matrici
   LaplacianoH=0
   H_xx=0.
   H_yy=0.

   !*************************************************************
   ! Calcolo il Laplaciano di H
   do iy=2,my
      do ix=2,mx

         H_xx(ix,iy)=(S(ix+1,iy)-2.*S(ix,iy)+S(ix-1,iy))/(hx)**2
```

```
        enddo
    enddo

    do iy=2,my
        do ix=2,mx

            H_yy(ix,iy)=(S(ix,iy+1)-2.*S(ix,iy)+S(ix,iy-1))/(hy)**2
            LaplacianoH(ix,iy)=H_xx(ix,iy)+H_yy(ix,iy)
        enddo
    enddo

    LaplacianoH=-LaplacianoH
    !*****************************************************************



 !open(unit=21,file='risultato1.dat',action="write",status="replace")
 !write(21,*) deltah
 !close(21)

 !open(unit=22, file='verifica.dat',action="write",status="replace")
 !write(22,*) LaplacianoH
 !close(22)



open(unit=23, file='terminenoto.dat',action="write",status="replace")
    write(23,*) S11
    close(23)
    return
end
```

Appendice D

cyclone.f90

```
!*****************************************************************
!
!     June 2016
!
!     Solver for the shallow water equations over a beta-plane
!     To be used for the study of cyclone motion
!
!     Square domain with periodic boundary conditions
!
!*****************************************************************

program cyclone

!*******************************************************
  implicit none
  include 'params.h'
  include 'com.h'

  integer icount, istep, i, j, l
  integer itime, idt
```

```
!  Set input-output files

   call setio

!  Read input parameters (e.g., dt, f0, beta, etc...)

   call conini
!
   time = 0.0

   call initc

!*********************************************************
!  Runge-Kutta time-stepping

! main loop

   itime=0
   idt=dt

   do l=1,nsteps
   itime=l*idt
      call steprk
      if(mod(itime,43200).eq.0) then
         call outvort(itime)
      endif
   enddo

   call outwrit

   stop 'voila'
end program cyclone
```

```
!*****************************************************
!
subroutine setio
!
!*****************************************************

  open(unit=72,file='inp_cyclo.h',status='old')
  open(unit=66,file='cyclo.out',status='old',position='append')

  return
end subroutine setio

!*****************************************************

subroutine conini

!*****************************************************
  implicit none
  include 'params.h'
  include 'com.h'

!-----------------------------------------------------
! read the basic parameters of the run
!
!     DT  =  time step in seconds
!
!     NSTEPS  =  number of time steps in the simulation
!
!     NOUT    =  every NOUT time steps the solution is
                 written on the file CYCLO.out
!
!     DISSIP  =  coefficient of the hyperdiffusion term
```

```
!
!     NCONS   =  every NCONS time steps the values of the
!                global invariants are computed ( call GLINV )
!-------------------------------------------------------------------

  integer i,j

  read(72,*) ncase
  read(72,*) dt
  read(72,*) nsteps
  read(72,*) nout
  read(72,*) dissip

! define computational box: LX,LY half-leghts
  read(72,*) LX
  read(72,*) LY

  ! define central value of f
  read(72,*) f0

  ! define beta
  read(72,*) beta
!      print*,'dt     = ',dt

!! gravitational acceleration
  grav=9.822

!!! bisogna definire tutte le quantita' che leggeremo
!!! da inp_cyclo prima di cominciare il time-stepping,
!!! e che "definiscono" il run

!  grid spacings in the x and y directions
  dx   = 2.*LX/float(nx-1)
```

```
  dy    = 2.*LY/float(ny-1)
  ! beta-plane; left low corner
  x0=-LX
  y0=-LY
!      print*,'dx = ',dx

  dissip1 = dissip

!  coefficients of numerical derivatives

  if(ns.eq.2) then
     cder(1) =   8. / (12.*dx)
     cder(2) = - 1. / (12.*dx)
     cder(3) =   0.0
  endif
  if(ns.eq.3) then
     cder(1) =  (3./2.) *(1./(2.*dx))
     cder(2) = -(3./10.)*(1./(2.*dx))
     cder(3) =  (1./30.)*(1./(2.*dx))
  endif

!  coefficients for the Adams-Bashforth scheme:

  ab1 = dt*23./12.
  ab2 = dt*16./12.
  ab3 = dt*5./12.


!  coefficients for the hyperdiffusione term
  dif(1)=-dissip1
  dif(2)=-dissip1
  dif(3)=-0.2*dissip1
```

```
!  compute Coriolis parameter f = f0 + beta*y

  do j=1,ny
     do i=1,nx
        f(i,j) = f0 + beta*(y0+(j-1)*dy)
     enddo
  enddo
!  write the input parameters

  return
end subroutine conini


!*********************************************

subroutine steprk

!*********************************************

!  Perform a fourth order Runge-Kutta time-step
!---------------------------------------------------------------------

  implicit none
  include 'params.h'
  include 'com.h'

  integer i, j ,l
  real zdt2, zdt3,zdt6
  real zrk(nx,ny,3), zsol0(nx,ny,3), zsol(nx,ny,3)

  zdt2 = dt/2.
  zdt3 = dt/3.
  zdt6 = dt/6.
```

```
!print*,' in RK ',zdt2,zdt3,zdt6
  zsol0 = sol

!  first R-K step

  call rhs(zrk)

  do l= 1,3
     do j=1,ny
        do i=1,nx
           zsol(i,j,l) = zsol0(i,j,l) + zdt6*zrk(i,j,l)
           sol(i,j,l)  = zsol0(i,j,l) + zdt2*zrk(i,j,l)
        enddo
     enddo
  enddo

  call rhs(zrk)
  do l= 1,3
     do j=1,ny
        do i=1,nx
           zsol(i,j,l) =  zsol(i,j,l) + zdt3*zrk(i,j,l)
           sol(i,j,l)  = zsol0(i,j,l) + zdt2*zrk(i,j,l)
        enddo
     enddo
  enddo

  call rhs(zrk)
  do l= 1,3
     do j=1,ny
        do i=1,nx
           zsol(i,j,l) = zsol(i,j,l) + zdt3*zrk(i,j,l)
           sol(i,j,l)  = zsol0(i,j,l) + dt*zrk(i,j,l)
```

```
        enddo
      enddo
    enddo

    call rhs(zrk)
    do l= 1,3
      do j=1,ny
        do i=1,nx
          sol(i,j,l) = zsol(i,j,l) + zdt6*zrk(i,j,l)
        enddo
      enddo
    enddo

    return
end subroutine steprk


!*****************************************************

subroutine rhs(zrk)

!*****************************************************
    implicit none
    include 'params.h'
    include 'com.h'

    integer i,j
    real zrk(nx,ny,3), solx(nx,ny,3), soly(nx,ny,3)
    real zu, zv, hypdiff(nx,ny,3)
!
! call routine that computes x and y derivatives of
! sol, and the hyperdiffusion terms
    call deriva(solx,soly,hypdiff)
```

```
!
  do j=1,ny
     do i=1,nx

        zu = sol(i,j,1)
        zv = sol(i,j,2)

        zrk(i,j,1) = - ( zu*solx(i,j,1) + zv*soly(i,j,1))&
             - solx(i,j,3) + f(i,j)*sol(i,j,2)       &
             + hypdiff(i,j,1)
        zrk(i,j,2) = - ( zu*solx(i,j,2) + zv*soly(i,j,2))&
             - soly(i,j,3) - f(i,j)*sol(i,j,1)       &
             + hypdiff(i,j,2)
        zrk(i,j,3) = - ( zu*solx(i,j,3) + zv*soly(i,j,3))&
             - sol(i,j,3)*(solx(i,j,1) + soly(i,j,2)) &
             + hypdiff(i,j,3)
     enddo
  enddo

  return
end subroutine rhs


!*************************************************************
!
subroutine deriva(solx,soly,hypdiff)
!
!*************************************************************
  implicit none
  include 'params.h'
  include 'com.h'

  integer i,j,l
```

```
  real solx(nx,ny,3), soly(nx,ny,3), hypdiff(nx,ny,3)
!
  real solex(nxt,nyt,3)

! define extended sol matrix
  solex=0.0

  do l=1,3
     do j=jmin,jmax
        do i=imin,imax
           solex(i,j,l) =sol(i-ns,j-ns,l)
        enddo
     enddo
! stencils for x derivatives
     do j=jmin,jmax
        do i=imax+1,imax+ns
           solex(i,j,l) =sol(i-imax+1,j-ns,l)
        enddo
        do i=1,ns
           solex(i,j,l) =sol(nx-ns-1+i,j-ns,l)
        enddo
     enddo
     ! stencils for y derivatives
     do i=imin,imax
        do j=jmax+1,jmax+ns
           solex(i,j,l) =sol(i-ns,j-jmax+1,l)
        enddo
        do j=1,ns
           solex(i,j,l) =sol(i-ns,ny-ns-1+j,l)
        enddo
     enddo
  enddo
```

```
!  compute first derivatives of u, v, and gh

  if(ns.eq.2) then

!!**** derivative *****
      do l=1,3
         do j=jmin,jmax
            do i=imin,imax
               solx(i-ns,j-ns,l) = cder(1)*(solex(i+1,j,l) - &
                    solex(i-1,j,l))+ cder(2)*(solex(i+2,j,l) &
                    - solex(i-2,j,l))
               soly(i-ns,j-ns,l) = cder(1)*(solex(i,j+1,l) -&
                    solex(i,j-1,l))+ cder(2)*(solex(i,j+2,l)-&
                    solex(i,j-2,l))
            enddo
         enddo
      enddo
  endif

!  Computee hyperdiffusion terms

  if(dissip.ne.0.0) then
       do l=1,3
         do j=jmin,jmax
            do i=imin,imax
               hypdiff(i-ns,j-ns,l)  = dif(l)*    &
                  (   12.*solex(i,j,l)
&
                - 4.*(   solex(i+1,j,l) + solex(i-1,j,l)&
                + solex(i,j+1,l) + solex(i,j-1,l) )     &
                + solex(i+2,j,l) + solex(i-2,j,l)       &
                + solex(i,j+2,l) + solex(i,j-2,l) )
            enddo
```

```
        enddo
      enddo
  endif
  return
end subroutine deriva

!*******************************************************
!
subroutine initc
!
!*******************************************************
  implicit none
  include 'params.h'
  include 'com.h'
! ******* initialization for pure advection test *******
! inizialization

  real alfa, u0(nx,ny), v0(nx,ny),deltah(nx,ny)
  real zy,cost
  integer i, j
  real pi
! def. pi
  pi=4.*atan(1.0)
!! ncase = 1: steady state
!*******************************************************
  if (ncase.eq.1) then

      alfa=pi/LY
      u_0=5.
      h_sud=5000.
      gamma=1.
      cost=f0*LY*u_0/(pi*grav)
      do j=1,ny
```

```
          zy=-LY+(j-1)*dy
          do i=1,nx

             sol(i,j,1)=u_0*cos(alfa*gamma*zy)
             sol(i,j,2)=0.
!               sol(i,j,3)=h_sud+ cost*sin(alfa*gamma*zy)
             sol(i,j,3)=h_sud - cost*sin(alfa*gamma*zy)
! multiply by g
             sol(i,j,3)=grav*sol(i,j,3)
          enddo
       enddo
!*******************************************************
  else
!*******************************************************
! read initial conditions
       open(unit=18, file='u.dat',status='old',action='read')
       read(18,*) u0
       close(18)

       open(unit=19, file='v.dat',status='old',action='read')
       read(19,*) v0
       close(19)

       open(unit=20,file='deltah.dat',status='old',action='read')
       read(20,*) deltah
       close(20)

       do j=1,ny
          do i=1,nx
             sol(i,j,1)=u0(i,j)
             sol(i,j,2)=v0(i,j)
! define sol(3), adding H to the height anomaly and multiplying by g
             sol(i,j,3)=grav*(10000.+deltah(i,j))
```

```
          enddo
        enddo
    endif
!
    sol_0=sol
!*****************************************************************
! memorizzo il flow iniziale nel file risultato.dat
open(unit=24,file='profilo_iniziale.dat',action="write",&
         status="replace")
    do j=1,ny
        do i=1,nx
            write(24,*) sol_0(i,j,3)
        enddo
    enddo
    close(24)

    return
end subroutine initc

!*****************************************************************
!
subroutine outvort(itime)
!
!*****************************************************************
    implicit none
    include 'params.h'
    include 'com.h'

    integer i, j, itime,iind, jind, kcall
    real hypdiff(nx,ny,3), vort(nx,ny), solx(nx,ny,3), soly(nx,ny,3)
    real xmax(6),ymax(6),zmax,d1,d2,v1,v2,d3,v3
    !!*************************************************************
```

```
  print*,' in outvort, time = ',itime

  kcall=itime/43200

! compute vorticity
  call deriva(solx,soly,hypdiff)

  do j=1,ny
     do i=1,nx
        vort(i,j)= solx(i,j,2)-soly(i,j,1)
     enddo
  enddo

! compute location of vorticity maximum

  zmax=0.
  iind=1
  jind=1

  do j=1,ny
     do i=1,nx
        if(vort(i,j).gt.zmax) then
           zmax=vort(i,j)
           iind=i
           jind=j
        endif
     enddo
  enddo
  !define location of max
  xmax(kcall)=-LX+(iind-1)*dx
  ymax(kcall)=-LY+(jind-1)*dy

  print*,' (i,j) max vort = =',iind,jind
```

```
  print*,' (x,y) max vort = =',xmax(kcall),ymax(kcall)

! at last call, compute average vortex speed over the last day

  if(kcall.eq.6) then
     d1=sqrt((xmax(6)-xmax(5))**2+(ymax(6)-ymax(5))**2)
     v1=d1/43200.
     d2=sqrt((xmax(5)-xmax(4))**2+(ymax(5)-ymax(4))**2)
     v2=d2/43200.
!average
     v1=(v1+v2)/2.

!print final vorticity
     print*,' vortex speed = ',v1
     open(unit=28, file='vort_finale.dat',action="write")
     do j=1,ny
        do i=1,nx
           write(28,*) vort(i,j)
        enddo
     enddo
     close(28)
  endif

  return
end subroutine outvort


!*****************************************************************
!
subroutine outwrit
!
!*****************************************************************
  implicit none
```

```
      include 'params.h'
      include 'com.h'

      integer i, j
      real hypdiff(nx,ny,3), vort(nx,ny), solx(nx,ny,3), soly(nx,ny,3)

      write(66,*) '************  OUTPUT FROM test.f90 ************'
      write(66,*)
      write(66,950) nx, ny, ns
      write(66,951) dx, dy, dt
      write(66,952) ncase, nsteps, nout
      write(66,953) dissip
      if(ncase.eq.1) then
         write(66,*) '------- GLOBAL STEADY-STATE TEST CASE --------- '
         write(66,*)
         write(66,954) LX, u_0, h_sud
      endif

!     write solution at t=nsteps*dt
      open(unit=25, file='h_finale.dat',action="write")
      do j=1,ny
         do i=1,nx
            write(25,*) sol(i,j,3)
         enddo
      enddo
      close(25)

      open(unit=26, file='u_finale.dat',action="write")
      do j=1,ny
         do i=1,nx
            write(26,*) sol(i,j,1)
         enddo
      enddo
```

```
  close(26)

  open(unit=27, file='v_finale.dat',action="write")
  do j=1,ny
     do i=1,nx
        write(27,*) sol(i,j,2)
     enddo
  enddo
  close(27)


950    format(1x,'nx = ',i4,3x,'ny = ',i4,3x,'ns = ',i4,/)
951    format(1x,'dx = ',1pe9.2,3x,'dy = ',&
              1pe9.2,3x,'deltat = ',1pe9.2,/)

952    format(1x,'ncase = ',i1,3x,'nsteps = ',&
              i6,3x,'nout = ',i6,/)
953    format(1x,'dissip = ',1pe9.2,/)
954    format(1x,'LX= ',1pe9.2,3x,' u0 = ',1pe9.2,&
              3x,'h0 = ',1pe9.2,/)

  return
end subroutine outwrit


!*****************************************************************
!
subroutine outwrit_errors
!
!*****************************************************************
  implicit none
  include 'params.h'
  include 'com.h'
```

```
  write(66,*)
!  write(66,970) time
  write(66,971) appo
  write(66,972)
  write(66,973) errl2gh
  write(66,974)
  write(66,973) errl2V

972    format(1x,'L2 error on gh ')
971    format(1x,'Errore calcolato dopo ',i6,' giorni')
973    format(1x,1pe10.3,1x,/)
974    format(1x,'L2 error on V ')

  return
end subroutine outwrit_errors
```

com.h

```
integer nsteps, nout, ncase,appo

real dx, dy, dt, x0, y0, time, dissip,&
      dissip1, ab1, ab2, ab3, LX, LY
real sol, f, rhsN, rhsNm1, rhsNm2, cder, &
     f0, beta, dif, sol_0
real u_0, h_sud, gamma, errl2gh, errl2V, grav

common /scal/ dx, dy, dt, x0, y0, time,&
       nsteps, nout, dissip, dissip1, ab1,&
        ab2, ab3, LX, LY,f0, grav, ncase,appo

common /vect/sol(nx,ny,3),f(nx,ny), cder(3),&
         dif(3),rhsN(nx,ny,3), rhsNm1(nx,ny,3), &
         rhsNm2(nx,ny,3),sol_0(nx,ny,3)
```

```
common /steady/u_0, h_sud, gamma, errl2gh, errl2V
```

params.h

```
integer nx, ny, ns, nxt, nyt, imin, imax, jmin, jmax
  parameter(nx= 401)
  parameter(ny=401)
  parameter(ns= 2)
! parameter(nw = 2*ns, nwh = ns)
  parameter(nxt = nx+2*ns)
  parameter(nyt = ny+2*ns)
! parameter(nlmax = ns*(ny + 2*no))
  parameter(imin = ns+1, imax = nxt-ns, jmin = ns+1, jmax = nyt-ns)
```

inp_cyclo.h

```
2
20.
12960
100
7.e-6
4.e6
4.e6
0.5e-4
2.15e-11
```

Bibliografia

[1] C.G. ROSSBY. *Relation between variations in the intensity of the zonal circulation of the atmosphere and the displacements of the semi-permanent center of action*, J. Marine Res. (1939).

[2] C.G. ROSSBY. *On displacements and intensity changes of atmospheric vortices*, J. Marine Res. (1948).

[3] JOHNNY C. L. CHAN AND R. T. WILLIAMS *Analytical and Numerical Studies of the Beta-Effect in Tropical Cyclone Motion. Part I:Zero Mean Flow* , Departement of Meteorology, Naval Postgraduate School, Monterey, (1985-1986).

[4] BENOIT CUSHMAN-ROISIN AND JEAN-MARIE BECKERS *Introduction to Geophysical Fluid Dynamics Physical and Numerical Aspects*,

[5] JOHNNY C. L. CHAN *The physics of tropical cyclone motion*, Annu. Rev. Fluid Mech. 2005. 37:99–128.

[6] DAVID L. WILLIAMSON *A Standard Test Set for Numerical Approximations to the Shallow Water equation in Spherical Geometry*, Journal of computational physics 102 (1992) .

[7] RONALD. B. SMITH *A Hurrican Beta-Drift Law*, Notes and Correspondence, Department of Geology and Geophysics, Yale University (1993).

[8] JAMES R. HOLTON *An Introduction to Dynamic Meteorology*,