ATLANTIC AREA
Transnational Programme
INVESTING IN OUR COMMON FUTURE

European Union
European Regional Development Fund

ARCOPOL

Development of OILTRANS Model code

Activity 4

Task 4.3: Drift and Pollutants Behaviour Predictions

ARCOPOL

The Atlantic Regions' Coastal Pollution Response

Version:	1.0
Last updated on:	23/12/2011
Author:	Alan Berry
Responsible partner:	IST
Involved partners:	Marine Institute

OVERVIEW

The **OIL TRANS**port Lagrangian model (OILTRANS) is an off-line particle-tracking model that runs with the stored predictions of a both a 3D hydrodynamic model, specifically the Regional Ocean Modelling System (ROMS) and a wave model, specifically SWAN.

OILTRANS is based on the LTRANS code base which was built to simulate oyster larvae. The LTRANS model was designed to predict the movement of particles based on advection, turbulence and larval behaviour.

It includes a 4th order Runge-Kutta scheme for particle advection and a random displacement model for vertical turbulent particle motion. The original LTRANS code was built by Elizabeth North and Zachary Schlag of University of Maryland Center for Environmental Science Horn Point Laboratory. It was written in Fortran 90 and is designed to track the trajectories of particles in three dimensions

Components of LTRANS have been in development since 2002 and are described in the following publications: North et al. 2005, North et al. 2006a, North et al. 2006b, and North et al. 2008.

The LTRANS code was adapted to simulate the mechanical spreading and physical fate processes of oil particles. OILTRANS expanded the processes governing particle transport to include; mechanical spreading of oil slicks, wind drift, stokes drift, langmuir circulation and shoreline beaching.

The code has an external and internal time step and boundary condition algorithms that keep particles from leaving the model domain. The external time step is the time step of hydrodynamic model output (e.g., 1hr). The internal time step is the time interval during which particle movement is calculated (e.g., 120 sec). The internal time step is smaller than the external time step so that particles do not move in large jumps that could cause inconsistencies between predictions of the hydrodynamic model and the particle tracking model.

At each internal time step of the transport model, particle motion is calculated as the sum of movement due to advection, turbulence, wind drift, stokes drift, mechanical spreading, langmuir circulation along with the original LTRANS options for larval behaviour. The model contains sub-models for each of these components. The transport and behaviour routines can be turned off so that particle movement is based solely on advection.

INTERPOLATION SCHEME

The ROMS and SWAN model predictions (stored in NetCDF format) are read in and interpolated in space and time to the particle location. The first step in the process of interpolating the wave and water properties (e.g., current velocities, salinity, temperature, sea surface height, and vertical and horizontal diffusivities, wave period, wave height, wind speed) to the particle location is to determine the grid cell in which the particle is located. For this, the ‘crossings’ point-in-polygon approach coupled with a search algorithm for computational efficiency is used.

Once the particle is located in a grid cell, water properties are interpolated in space to the particle location. All water properties are interpolated from the native ROMS and SWAN grid points (i.e., u grid points are used to calculate *u*-velocity at the particle location, v grid points are used for *v*-velocity, and rho grid points are used for sea surface height, *w*-velocity, salinity, and diffusivity calculations).

For two-dimensional water properties (e.g., sea surface height, water depth, wave height, etc) bilinear interpolation is used. For three-dimensional water properties (e.g., current velocities, diffusivities, salinity), a water-column profile scheme is applied (North et al. 2006a). In this scheme, values are interpolated along each s-level to create a vertical profile of values at the x-y particle location.

A tension spline curve is then fit to the vertical profile and used to estimate the water property at the particle location. The interpolation scheme was adapted from North et al. (2006a), streamlined to increase computational speed, and enhanced to handle model domains with irregular bottoms and non-rectangular grid geometries. It should be noted that this interpolation scheme likely assumes that the underlying hydrodynamic model grid is orthogonal (Rich Signell, pers. comm.).

Although there are several available methods for interpolating to the particle location (e.g., linear interpolation, cubic splines) a sophisticated tension spline curve fitting routine is used. Both cubic and simple tension splines cause 'offshoots'. Offshoots occur when the interpolated line does not preserve the monotonicity and concavity of the original data.

For particle tracking, it is necessary to interpolate in time as well as space because the duration between successive outputs of the hydrodynamic models (i.e., the external time step) is longer than the time step of particle motion (i.e., the internal time step). To do this, water properties are estimated at the particle location (as above) at three time points (previous, current and future) that correspond to the hydrodynamic model output. Then a polynomial curve is fit to the water properties at three time points and used to calculate the water properties at the time of particle motion (i.e. for the internal time step).

Advection sub-model.

A 4^{th} order Runge-Kutta scheme in space and time is used to calculate particle movement due to advection. This scheme solves for the *u*-, *v*-, and *w*- current velocities (representing the x-, y-, and z-directions) at the particle location using an iterative process that incorporates velocities at previous and future times to provide the most robust estimate of the trajectory of particle motion in water bodies with complex fronts and eddy fields like Chesapeake Bay.

Current velocities (m s^{-1}) provided by the Runge-Kutta scheme are multiplied by the duration of the internal time step (δt) to calculate the displacement of the particle in each component direction. Displacements (m) are then added to the original location of the particle (x_n, y_n, z_n) in order to calculate the new location of the particle (x_{n+1}, y_{n+1}, z_{n+1}):

$$(1) \quad x_{n+1} = x_n + u\delta t$$

$$(2) \quad y_{n+1} = y_n + v\delta t$$

$$(3) \quad z_{n+1} = z_n + w\delta t$$

The *u* and *v* current velocities are separated into north and east component directions before particle motion is estimated. Law-of-the-wall (a log layer calculation) is applied to the current velocities within one s-level of bottom to simulate reduction in current velocities near bottom.

NOTE:
The LTRANS model was designed to maintain fidelity with hydrodynamic model predictions. All interpolation occurs from the original staggered grid of the u, v, and rho grid points directly to the particle location. In addition, horizontal interpolation occurs along s-levels in an attempt to follow the structure of the hydrodynamic model in regions of changing bathymetry. These interpolation schemes may be costly in computation time compared to less accurate schemes; the benefits have not been quantified. The LTRANS model was developed to simulate oyster larvae in Chesapeake Bay, a region with complex bathymetry and horizontal and vertical current shears. It is not known whether the LTRANS interpolation schemes would be appropriate in other systems, and, if so, in what conditions they should be used.

INPUT FILES

Model Grid and bathymetry

The OILTRANS model uses hydrodynamic data from ROMS NetCDF files. It uses two files, a bathymetry grid file that contains information about the model grid, and the output files that contain the hydrodynamic model predictions.

The following variables should be in the bathymetry grid file that contains the ROMS model grid information:

angle	angle between x-coordinate and true east direction
h	depths of rho nodes
mask_rho	rho node mask value
mask_u	u node mask value
mask_v	v node mask value
lon_rho	longitude coordinates of rho nodes
lon _u	longitude coordinates of u nodes
lon _v	longitude coordinates of v nodes
lat_rho	latitude coordinates of rho nodes
lat _u	latitude coordinates of u nodes
lat _v	latitude coordinates of v nodes

The main structure of OILTRANS is based on the assignment of a unique number to each ROMS model grid point (referred to as a node). Each grid cell (referred to as an 'element') is comprised of a set of 4 nodes. After the hydrodynamic data is read from the NetCDF files into the variables listed

above, it is reorganized so that each data point is assigned the appropriate node number. This is done in the subroutine **initGrid** in the Hydrodynamic Module after the grid variables are read in.

Model archived hydrodynamics

The following variables should be in the ROMS output files that contain the archived hydrodynamic model predictions. OILTRANS assumes that the sequential ROMS output files contain the same number of time steps in each file (e.g., if the first file contains predictions at 24 discrete times, then all files should contain predictions at 24 discrete times).

Note: Variables **Sc_r**, **Sc_w**, **Cs_r**, and **Cs_w** must be in the first output file used by OILTRANS. The other variables should be in all of the output files used by OILTRANS.

Sc_r s coordinate on rho grid
Cs_r Cs value on rho grid
Sc_w s coordinate on w-grid
Cs_w Cs value on w-grid

Aks vertical diffusivity of salinity at rho nodes
salt rho node salinity
temp rho node temperature
u u-direction velocity
v v-direction velocity
w w-direction velocity
dens rho node density
zeta zeta levels at rho nodes

SWAN input files

The OILTRANS model has the option to use wind and wave data from SWAN model NetCDF files. OILTRANS assumes that the sequential SWAN output files contain the same number of time steps in each file (e.g., if the first file contains predictions at 24 discrete times, then all files should contain predictions at 24 discrete times). The following variables should be in the SWAN output files that contain the archived wind and wave model predictions.

Note: The SWAN model must generate model predictions on the same model grid as the ROMS model.

Hs significant wave height
tm_01 mean wave period
u10 10m wind speed U component
v10 10m wind speed V component
Pd peak wave direction
Pwl peak wave length

Particle locations input file

LTRANS:

The particle locations are read in from a .csv file which contains either three or four columns: longitude, latitude, depth (in meters) and, if settlement is turned on, the id of the habitat polygon the particle starts on. This file must have at least as many rows as the number of particles in the parameter numpar. All of the particle start locations should be within the model boundaries.

OILTRANS:

The particle locations are read in from a .csv file which contains four columns: longitude, latitude, depth (in meters) and time of spill. The first entry in the .csv file defines the number of spill locations to be simulated. The file must have as many subsequent rows as the number of spill locations to be modelled (not the number of particles in the simulation). The particle start locations should be within the model boundaries.

Model parameters input file

One input file, OILTRANS_data, contains the parameters that are used to adapt OILTRANS to different ROMS hydrodynamic model domains, change particle attributes (e.g., turn on/off behaviour, oil module, etc), and set input/output file paths. All initialization variables are placed in this file so that the code does not need to be modified to run OILTRANS in different model domains or with different particle characteristics. Everything that the user may need to change can be found in OILTRANS_data, (see Appendix I).

MODEL EXECUTION

OILTRANS.f90 contains the main structure of the particle-tracking program. It executes the external time step, internal time step, and particle loops, advects particles, and writes output.

It calls the modules that read in hydrodynamic model information, move particles due to turbulence and (larval or oil) behaviour, weathers oil spills, test if particles are in habitat polygons, and apply boundary conditions to keep particles in the model domain.

Model Initialisation

Before the iterative loops that comprise the heart of the particle tracking model structure, OILTRANS.f90 starts with an initialisation subroutine called ini_LTRANS.

Subroutine **getParams** reads in the OILTRANS_data input file, making the parameters declared within available to all the other modules

subroutine **init_genrand** is called to initialise the Mersenne Twister random number generator, creating random numbers between 0 and 1 from a uniform distribution.

Several time stepping variables are calculated, variable arrays are initialized, and the particle locations are read in and their latitude and longitude coordinates are converted to meters.

If the OIL model is activated OILTRANS.f90 calls the subroutine **initOilModel**.that calculates the density and viscosity of the oil at the moment of the spill incident.

Subroutine **initGrid** is used to read the latitude and longitude coordinates of the nodes in the rho, u, and v grids, depth at the rho nodes, the angle between x coordinate and true east, masks of the rho, u, and v grid nodes that specify whether the nodes are on land or in water, and the variables necessary to calculate s-levels. It also assigns unique identification numbers to rho-, u- and v elements to create the OILTRANS grid element structure.

In addition, information about the ROMS hydrodynamic model domain is read in and used to create the OILTRANS model domain and grid element structure. In OILTRANS, an element is defined as a set of four adjacent rho, u or v nodes that form a quadrilateral. Each element is assigned a unique identification number. These numbers are used to store previous, and efficiently search for new, particle locations.

A number of subroutines are called to initialize the OILTRANS domain and element structure.

Subroutine **createBounds** defines the OILTRANS model boundaries based on the land/sea masking of the rho grid.

Subroutine **initBehave** is used to initialize the matrices that contain information on particle attributes for the Behavior Module of the original LTRANS code.

The code then does a series of checks on each particle to ensure it is within the model boundaries, and not within an island element of the model.

Finally, subroutine **initHydro** reads in the initial ROMS hydrodynamic data (u-, v-, and w-velocities, salinity, temperature, zeta, and vertical diffusivity) for the back, center, and forward time steps from the first sequential ROMS archived output file.

If the WindsWavesModel parameter is activated in the OILTRANS_data input file, the initial SWAN data (significant wave height, mean wave period, peak wave direction, peak wave length, 10m U and V wind speed components) for the back, centre, and forward time steps from the first SWAN sequential output file.

Control then passes back to the main OILTRANS.f90 code which calls the **run_LTRANS** subroutine, controlling the external and internal timestep loops, particle tracking and oil weathering procedures.

External Timestep Loop

This loop which iterates for each external time step contains the majority of the execution code of the program. The execution of the external time step loop can be broken down into three major sections: updating the hydrodynamic data, the internal time step loop, and the output (print) section. The internal time step loop will be covered in the following section.

The main purpose of the external time step loop is to update hydrodynamic data. The hydrodynamic data comes from ROMS NetCDF files and optionally SWAN netCDF files which contain information about u velocity, v velocity, w velocity, salinity, sea surface height, wave properties and wind speeds

To calculate water properties at the particle location, OILTRANS uses hydrodynamic model output from the current ('center') time step, the previous ('back') time step, and the future ('forward') time step. On the first iteration of the external time step the attributes of the back, center, and forward times are taken directly from the first netcdf file. However, on every subsequent iteration the back and center time steps' attributes are transferred from the previous center and forward time steps, respectively, and data from the netcdf files is only read in for the forward time step.

The duration of the external time step (in seconds) is set in OILTRANS_data with the variable **dt**. The value **dt** should be equal to the duration between the instantaneous snapshots of data in the hydrodynamic input files. For example, if one netcdf file contains 24 hrs of data stored at 1 hour intervals, then the external time step is 1 hour

The variable **tdim** found in OILTRANS_data should be initialized to the total number of external time steps within each hydrodynamic model output. The variable **stepT**, the total number of external time steps in the model, is **seconds** divided by **dt**, where **seconds** is the total number of seconds that the model will run.

The external time step consists of a loop from 1 to **stepT** using the variable **p** to iterate. The first two iterations use the same data, so the hydrodynamic data is initialized before the first iteration by calling subroutine **initHydro** and is not updated again until **p** is greater than 2.

On all other iterations, the program updates hydrodynamic data by calling subroutine **updateHydro**. In **updateHydro**, the 'forward' variables are updated with the most recent hydrodynamic data and the 'back' and 'center' variables are replaced with the 'center' and 'forward' variables from the previous time step, respectively.

Following the update hydrodynamic data section is a short section used to update the external time step values in **ex()**. The variable **ex()** is an array of three values used to store the back time, center time, and forward time in seconds. These values are calculated by using multiples of **dt**, the size of the external time step in seconds.

Internal Timestep Loop

The internal time step loop is the loop in which the particle tracking and oil weathering occurs. The internal time step is shorter than the external time step to allow particles to move in smaller intervals than the hydrodynamic model output intervals. Within each iteration of the internal time step loop, the time and internal time step values, **ix()**, are updated.

The OILTRANS model checks whether the correct **ix()** time has been reached to activate the oil spill. If so the model calls subroutine **InitialArea** which calculates the initial area of the spill after the gravity spreading phase has ceased, and the **PhaseTime** after which the gravity phase will have ceased.

After this, model goes into a loop from 1 to **numpar** through each particle, randomly distributing oil particles using a normal distribution throughout the theoretical Fay area of spill.

The model then checks if **PhaseTime** has been reached, in which case the oil weathering processes (evaporation, emulsification, dispersion, dissolution, density, viscosity, etc) are called depending on the options chosen in OILTRANS_data input file.

Once this is complete, the program enters the particle transport loop from 1 to **numpar** where particle movement due to advection, turbulence, winds and waves, is calculated over the time step. Then particle locations are updated. These events occur every iteration of the internal time step.

The duration of the internal time step, **idt**, must be set in OILTRANS_data. The variable **stepIT** (the number of internal time steps per external time step) is then initialized as the value of **dt** (the external time step) divided by **idt** (the internal time step).

The internal time step is a loop that iterates from 1 to **stepIT** using the variable **it**. The values of **ix()**, the internal time step values, are calculated. **Ix()** is an array with three values, so it can hold the internal 'back', 'center', and 'forward' times.

Model Termination

Once the internal and external timestep loops have been completed, control passes back to the OILTRANS.f90 program which calls the subroutine **fin_LTRANS**. The purpose of this subroutine is to write the final positions and status to the final output file, de-allocate local variables and module level variables and calculate model run time and output to screen before exiting.

Oil Module

The oil module was developed to predict the evolution and behaviour of the processes (transport, spreading and weathering) of oil spilled in the water. The processes included in the oil module are: spreading, evaporation, emulsification, dispersion, dissolution and oil beaching, along with the transport processes of wind drift, stokes drift and langmuir circulation.

The oil weathering module uses the ROMS archived hydrodynamics and optionally, the SWAN archived wind and wave fields. The trajectory of the oil slick is computed assuming that the oil can be idealised as a number of particles that independently move in the water. Except for oil spreading, dispersion and beaching, all weathering processes and properties are assumed uniform for all oil particles. Additionally, it is assumed that the temperature of the oil is the same as the ambient water temperature.

Different alternative methods were coded for the prediction of the oil spreading and evaporation processes. There is therefore more than one way of simulating the same process.

Weathering and movement processes of the oil slick can interact, with the weathering strongly influencing how the oil is moved. Weathering processes occur at very different rates, but all begin immediately after oil is spilled. Weathering rates are not constant throughout the duration of the spill, and are usually highest immediately after the spill. All weathering and transport processes are strongly dependant on the type of oil, the volume of oil spilled and the weather conditions during a spill event.

The order of importance of the various weathering processes are; evaporation, emulsification, dispersion, dissolution, photo-oxidation, sedimentation, and biodegradation. Only the first four weathering processes are included in the current OILTRANS model, as they account for 99% of the reduction in oil spill volume during the first week after a spill.

Spreading

Oil spreads horizontally over the water surface even in the complete absence of wind or water currents. The spreading is due to the force of gravity and the interfacial tension between oil and water. The oil viscosity opposes these forces. Usually within the first hour, the effect of gravity on the spreading of the oil slick is greatly reduced and the spreading of the slick is controlled based on the balance between the viscosity of the oil and the oil-water interfacial tension.

Most attempts at understanding the spreading process have produced formulas that only roughly approximate the actual spill results, as the spreading process is a complex interaction between the physical properties of the oil and the environmental state of the sea surface.

The most widely used formulations for determining the spread of oil on the water's surface are modified versions of the now classical equation proposed by Fay, 1969.

The current OILTRANS model incorporates four different formulations for the surface spreading of the oil slick. The first three formulations, ADIOS2, MOHID2 and CONCAW are similar implementations of a Fay algorithm, but with different parameterisations for radial spreading.

The fourth formulation, that of Lehr as used in OILPOL, attempts to account for the elongated spreading of the oil slick in a downwind direction using a modified version of the Fay spreading formulations.

Fay broke the spreading process into three phases; the first phase in which only gravity and inertia forces are important, the second phase in which gravity and viscous forces dominate and a final phase in which surface tension is balanced by viscous forces.

Initial Area of Spill

ADIOS2 & MOHID2

The first stage allows the oil to spread due to its gravitational potential and occurs rapidly (<1hr), even for large spill. The time to the end of this stage, **PhaseTime**, is calculated by MOHID2 and ADIOS2 as:

$$t_o = \left(\frac{k_2}{k_1}\right)^4 \left(\frac{V_o}{\upsilon_w g \Delta}\right)^{1/3}$$

where: k_1,k_2 = empirical coefficients whose value depend on the researcher
V_o = volume of oil spilled
υ_w = kinematic viscosity of water
g = gravitational acceleration
$\Delta = \frac{\rho_w - \rho_{oil}}{\rho_w}$ = relative density difference between the water and oil

OILTRANS assumes that during the gravity spreading phase none of the weathering processes are taking place. In essence, **PhaseTime**, t_o, is the starting time of the oil spill model. The **InitialArea,** A_o, at the end of this gravity spreading phase is calculated by ADIOS2 and MOHID2 to be:

$$A_o = \pi \frac{k_2^4}{k_1^2} \left(\frac{V_o^5 g \Delta}{\upsilon_w^2}\right)$$

For MOHID2 formulations, k_1 & k_2 = 1.14 & 1.45 respectively
For ADIOS2 formulations, k_1 & k_2 = 1.53 & 1.21 respectively

CONCAW

For the CONCAW implementation of the Fay formulation the spill areas for both the gravity spreading phase and the viscous spreading phase are calculated over time.

Transition from the gravity spreading phase to the viscous spreading phase is assumed to occur when the formulae predict spills of the same area. The time at which both formulae predict spills of the same area is **PhaseTime**. The method is presented below:

Do While Area1 <= Area 2

$Area1 = \pi.1.14^2.\sqrt{\Delta g V}.t$ = area of gravity spreading

$Area2 = \pi.0.98^2.\left(\frac{\Delta g V^2}{\sqrt{\upsilon_w}}\right)^{\frac{1}{3}}.\sqrt{t}$ = area of viscous spreading

t = t +10 increment time (in seconds)

End Do

PhaseTime = t
InitialArea = Area2

OILPOL

Development of OILTRANS Model code

The OILPOL model does not use the Fay spreading methodology for determining the time until gravity spreading phase is complete, rather, the **InitialArea,** A_o, of the spill is calculated as:

$$A_o = \pi.\left(2.81\sqrt{V_o}\right)^2.$$

Spreading of Spill

ADIOS2 & MOHID2

Both methods use the second phase of the Fay formulas, the so-called gravity viscous spreading. Fay predicted the area, A_t, of the slick over time to be described by:

$$A_t = \pi k_2^2 \left(\frac{V_o^2 g \Delta t^{3/2}}{\sqrt{\upsilon_w}} \right)^{1/3}$$

Both methods approximate the Fay spreading with a diffusion process, where the diffusion coefficients, D_x & D_y are given by:

$$D_x = D_y = \frac{k_2^2}{16} \left(\frac{V_o^2 g \Delta}{\sqrt{\upsilon_w}} \right)^{1/3} \frac{1}{\sqrt{t}}$$

In addition, in ADIOS2, a second spreading process designed to represent eddy diffusion of surface waters is added to the Fay diffusion coefficients. Based on experimental results, a time dependant diffusion parameter best represents the empirical results. The diffusion parameter, D_{eddy} is represented as:

$$D_{eddy} = 0.033\ t^{0.16}$$

The diffusion coefficients are then converted into uniformly distributed random velocities, u_r & v_r in the range $[-U_r, U_r]$ and $[-V_r, V_r]$ for each particle as detailed below.

The relationship between the diffusion coefficients and the velocity ranges $[-U_r, U_r]$ and $[-V_r, V_r]$ is expressed as:

$$U_r = \sqrt{\frac{2D_x}{\delta t}}\ ,\ V_r = \sqrt{\frac{2D_y}{\delta t}}$$

where: δt = timestep interval

Random velocities, u_r & v_r, (with a uniform distribution) inside the velocity ranges $[-U_r, U_r]$ and $[-V_r, V_r]$ are then assigned to each particle in the following way;

$$u_r = R_1 \cos(2\pi R_2).U_r$$
$$v_r = R_1 \sin(2\pi R_2).V_r$$

where: R_1 & R_2 are randomly generated numbers between 0 and 1.

An 'origin' particle which is not included in the Fay spreading processes is used as a reference point around which to diffuse all other oil particles with the random velocities $[-u_r, u_r]$, $[-v_r, v_r]$ at each time step by:

$$Px = Pox + (\pm)u_r$$
$$Py = Poy + (\pm)v_r$$

where : Px = particle x location
Py = particle y location
Pox = origin particle x location
Poy = origin particle y location

The 'origin' particle is free to advected by water currents, wind and stokes drift, thereby tracking the centroid of the spill.

CONCAW

The process used in the CONCAW methodology is to calculate the fractional increase in the radius of the theoretical area of the oil slick, CoefR, from one time step to another as:

$$CoefR = \frac{\sqrt{Area_T / \pi}}{\sqrt{Area_{T-1} / \pi}}$$

where: $Area_T$ = Area of slick at current timestep
$Area_{T-1}$ = Area of slick at previous timestep

The fractional increase in the radius of the slick is used to update each particles location with respect to the 'origin' particle location by:

$$Px_T = Pox_T + [(Px_{T-1} - Pox_{T-1}) * CoefR]$$
$$Py_T = Poy_T + [(Py_{T-1} - Poy_{T-1}) * CoefR]$$

where : Px = particle x location
Py = particle y location
Pox = origin particle x location
Poy = origin particle y location
$_T$ = current timestep
$_{T-1}$ = previous timestep

OILPOL

Development of OILTRANS Model code

In the OILPOL methodology the slick is assumed to spread as an ellipse with the major axis in the direction of the wind. The length of the minor axis, Q after the spreading starts is given by a Fay-like formula but with modified coefficients that were obtained by fitting to observed data, as:

$$Q = 1.13(\Delta V)^{1/3} t^{1/4}$$

The downwind axis of the ellipse, R, is given by:

$$R = Q + 0.0034 W^{4/3} t^{3/4}$$

where: W = wind speed (m/s)

The area of the slick, A_t at time t is therefore:

$$A = \frac{1}{4} \pi Q R$$

The process used in the OILPOL methodology is similar to that of the CONCAW methodology, namely, to calculate the fractional increase in the radius of the major and minor axes of the oil slick, CoefR & CoefQ respectively, from one time step to the next. The fractional increase in the major and minor axes of the slick are then used to update each particles location with respect to the 'origin' particle location in a manner similar to that outlined above in the CONCAW methodology.

Note: Currently OILTRANS assumes a constant wind direction for determining the spreading of the slick using this methodology.

Evaporation

In OILTRANS the evaporation process can be calculated using two different methods; one, evaporative exposure, proposed by Stiver and Mackay 1984, and one, a simplified empirical formulation, proposed by Fingas 1998.

Another method, the pseudo-component method of Jones 1997, as used by ADIOS2 requires a considerable amount of input data in relation to mean vapour pressure, solubility and molecular weight of each pseudo-component of the oil. Jones compared the pseudo-component approach to that of Fingas, and Stiver & Mackay and found for equivalent conditions, the pseudo-component method gave similar results to Fingas, and only slightly underpredicted results from Stiver & Mackay. Given the correlation between the pseudo-component method and the Fingas, and Stiver & Mackay methods, only the former methods have been encoded as they require less detailed information on oil type and constituents.

Stiver & Mackay

The evaporative exposure is given by the formula:

$$\frac{dVf_e}{dt} = \frac{K_e A_t}{V_o} \exp(6.3 - \frac{10.3}{T}(T_o + T_G Vf_e)$$

where: Vf_e = volume fraction of evaporated oil
K_e = mass transfer coefficient
A_t = Area of slick at time t.
V_o = volume of spill
T = oil temperature (assumed equal to water temperature)
T_o = initial boiling point of the oil
T_g = distillation curve gradient

The mass transfer coefficient, K_e is given by: $K_e = 1.5e^{-3}\ W^{0.78}$

The initial boiling point and the distillation curve gradient of the oil are determined from the API density of the oil according to the following relationship (for crude oil)

$$T_o = 532.98 - 3.1295\ API$$
$$T_g = 985.62 - 13.597\ API$$

Implementation of the evaporative exposure formulation in the OILTRANS code takes the form:

$$dVf_e = \frac{K_e A_t}{V_o} \exp(6.3 - \frac{10.3}{T}(T_o + T_G Vf_e)dt$$

with the volume fraction of oil evaporated at each timestep being calculated. After each timestep the value of F_e (total volume fraction of oil evaporated) is updated by dF_e.

The mass fraction of oil evaporated, $Mf_e = Vf_e \cdot \rho_{oil}$

Fingas

The other option to calculate the evaporation of oil was proposed by Fingas and is based on the variables of time and temperature. Fingas determined specific empirical equations for a wide variety of oil types of the form:

$$\%M_e = (\alpha + \beta T)\ln(t)$$

where: $\%M_e$ is the percentage (by weight) of evaporated oil
α,β are empirical constants depending on oil type
T is oil temperature (assumed equal to water temperature)
t is time after spill in minutes

From the range of experiments conducted, Fingas proposed the following general equation:

$$\%M_e = [0.165(\%D) + 0.045(T-15)]\ln(t)$$

where: %D is the percentage by weight distilled at 180°C.

The volume fraction of oil evaporated, $\%V_e = \%M_e / \rho_{oil}$

Emulsification

Emulsification is the process by which water droplets are dispersed into the oil slick. When water-in-oil emulsions form the physical properties and characteristics of the oil slick change dramatically.

Stable emulsions typically contain 60%-90% water, expanding the volume of the oil slick by 2 to 5 times the original volume. Most significantly, the viscosity of the oil changes from typically a few hundred mPas to about 100,000 mPas.

The formulation for the formation of water-in-oil emulsions in the OILTRANS code is based on the most recent work of Fingas, 2011, and is based on the oil density, viscosity, asphaltene, resin and saturates content of the oil.

The formulation defines the class of emulsion that may be formed; stable, mesostable, entrained or unstable, and assigns values to increases in viscosity based on the class of emulsion formed.

The stability class of emulsion that would be formed is calculated as follows:

$$\begin{aligned}\text{Stability Class} = {} & 12.3 + 0.259St - 1.601Rt - 17.2(A/Rt) \\ & - 0.50Vt^3 + 0.002Rt^3 + 0.001At^3 + 8.51(A/Rt)^3 \\ & - 1.12\ln(Vt) + 0.7\ln(Rt) + 2.97\ln(A/Rt) \\ & + 6e\text{-}8(\text{Exp}(Vt))^2 - 1.96(\text{Exp}(A/Rt))^2 \\ & - 4e\text{-}6\log(Dt)/(Dt)^2 - 1.5e\text{-}4\log(A/Rt)/(A/Rt)^2\end{aligned}$$

where: St = transformed saturate content
Rt = transformed resin content
A/Rt = transformed asphaltene/resin ratio
Vt = transformed natural logarithm of viscosity
At = transformed asphaltene content
Dt = transformed exponential of the oil density

The values of Stability Class assigned to each class of emulsion are given in the table below:

Table 1: Conditions for Emulsion Type Calculations

Calculated Stability Class minimum	maximum	Conditions	State	Error (%)
2.2	15		Stable	0
-12	-0.7		Mesostable	9
-18.3	-9.1	density >0.96 viscosity >6000	Entrained	7
-7.1	-39.1	density <0.85 or >1.0	Unstable	10

		viscosity <100 or >800000 asphaltenes or resins <1%		

The viscosity of the resulting emulsion can be taken as the average of the types at a given time as shown in the table below:

Table 2: Viscosity Increase from Starting Oil Viscosity

Emulsion Type	Viscosity Increase On First Day	Week	Year
Entrained	1.9	1.9	2.1
Mesostable	7.2	11	32
Stable	405	1054	991
Unstable	0.99	1.0	1.0

The kinetics of emulsion formation have also been studied by Fingas 2011, and data are available to compute the time to formation of the various emulsion types.

Application of the equations in the table below provide the time to formation of a particular water-in-oil emulsion, for a given wave height.

Table 3: Time to Formation predictor from Wave Height

Resulting Equation Predictor	Equation $Y = A + B / X^{1.5}$ A	B	R^2
Stable	27.1	7520	0.51
Mesostable	47	49100	0.95
Entrained	30.8	18300	0.94
X = wave height in cm			
Y = time to formation in minutes			

Dispersion

Dispersion, or entrainment, occurs when fine droplets of oil are transferred into the water column by wave action or turbulence. Large droplets (>70µm) tend to rise and will not stay in the water column for more than a few seconds.

The dispersion process is based on the classic method of Delvigne and Sweeney 1988 who developed a relationship for entrainment rate, Q_d, as a function of droplet size and oil viscosity, as:

$$Q_d = C^* D_d^{0.57} SFd^{0.7} \Delta d$$

where: Q_d is the entrainment rate, (kg/m^{2}s), for droplet diameter d, (m).
C* is an empirical entrainment constant which depends on oil type and weathering state.
D_d is the dissipated breaking wave energy per unit surface area, (J/m^2)
S is the fraction of sea surface covered by oil

F is the fraction of sea surface hit by breaking waves
d is the droplet diameter
Δd is the oil particle interval diameter, (m)

C*, the entrainment constant, was fitted to a series of experimental data according to the following relationships:

$$\text{If } (\mu/\rho_o) < 132 \text{ } (cSt), \; C^* = \exp[-0.1023 \ln(\mu/\rho_o) + 7.575]$$
$$\text{If } (\mu/\rho_o) \geq 132 \text{ } (cSt), \; C^* = \exp[-1.8927 \ln(\mu/\rho_o) + 16.313]$$

where: μ is the viscosity of the oil, (mPa/s)
ρ_o is the density of the oil, (g/cm^3)

D_d, the dissipated wave energy, is given by:

$$D_d = 0.0034 \, \rho_w g H_{break}^2$$

where: ρ_w is the density of seawater (kg/m3)
g is acceleration due to gravity
H_{break} is the rms of breaking wave height (m)

Two methods exist within OILTRANS to calculate the rms of the breaking wave height, H_{break}. If the **windwavesmodel** option is enabled, the breaking wave height is obtained from the SWAN model output(s). If the **windwavesmodel** is not enabled, then the breaking wave height is calculated according to the formula from the CERC Shore Protection Manual as;

$$H_{break} = \left(\frac{1}{\sqrt{2}}\right).\left(\frac{0.243.W_{10}^{\,2}}{g}\right)$$

where: W_{10} is the wind speed at 10m above sea surface

S, the fraction of the sea surface covered by oil, is assumed as unity. (1.0)

F, the fraction of sea surface hit by breaking waves per unit time, is parameterised as follows:

$$\text{if } W_{10} < W_{th}, \; F = 3e^{-6}(W_{10}^{\,3.5} / T_w)$$
$$\text{if } W_{10} > W_{th}, \; F = 0.032[(W_{10} - W_{th}) / T_w]$$

where: W_{th} is the threshold windspeed for onset of breaking waves (~ 6m/s)
T_w is the significant wave period (s)

Two methods exist within OILTRANS to calculate the significant wave period, T_w. If the **windwavesmodel** option is enabled, the significant wave period is obtained from the SWAN model output(s). If the **windwavesmodel** is not enabled, then the significant wave period is calculated according to the formula from the CERC Shore Protection Manual as;

$$T_w = 8.13.\left(\frac{W_{10}}{g}\right)$$

Δd, the oil particle interval diameter, is based on the mean droplet diameter, d_{50}. The mean droplet diameter d_{50} was curve fitted to data by Delvigne and Sweeney to form the following relationship:

$$d_{50} = 1818\ E^{-0.5}\ (\mu/\rho_o)^{0.34}$$

where: E is the wave energy dissipation rate per unit volume, (J/m^3s), and set as $1e^{-3}$ for breaking waves.

The minimum droplet diameter, D_{min}, is assumed to be 10% of the d50 value because volumes below this size are relatively small and can be neglected.

The maximum droplet diameter, D_{max}, is set to equal the mean, because in testing droplets larger than d50 were found to resurface in less than one timestep, and so are not different to the surface slick.

The oil particle interval diameter is then constructed by adopting 5No. size classes between D_{min} and D_{max}, equally spaced on diameter, as:

$$\Delta d = (D_{max} - D_{min}) / 5.0$$

Therefore, the total entrainment rate, Q_{total} (kg/m^2s), for all droplet size classes is:

$$Q_{total} = \sum_{i=1}^{5} C^* D_d^{0.57} SFd_i^{0.7} \Delta d$$

And the total mass entrained, M_{ent} (kg), per time step is equal to:

$$M_{ent} = Q_{total} * A_t * \delta t$$

where: A_t is the area of slick at time t
δt is the timestep interval

Density

The initial oil density is obtained from either the API gravity of the oil, or the density value and reference temperature, both contained within the oil database. Only oils with densities lower than water are modelled, as more dense oils will sink.

The change in oil density over time is related to three different processes; changing water temperature, evaporation, and emulsification.

The change in density due to changing temperature can be expressed as:

$$\rho_{oil} = \rho_{ref}\left(1.0 - CDensT\left(T - T_{ref}\right)\right)$$

where: ρ_{oil} is density of oil for given water temperature, T
ρ_{ref} is original (reference) density of oil at reference temperature T_{ref}
CDensT is an empirical constant (= 8e-4 (ref: ADIOS2))

The change in density due to evaporation can be expressed as:

$$\rho_{oil} = \rho_{ref}\left(1.0 + CDensE.F_{evap}\right)$$

where: CDensE is an empirical constant (= 0.18 (ref: ADIOS2))
F_{evap} is the fraction of oil evaporated from the slick

The change in density due to emulsification can be expressed as:

$$\rho_{oil} = \left(Y\rho_w\right) + \rho_{ref}\left(1.0 - Y\right)$$

where: Y is the water content of the water-in-oil emulsion
ρ_w is the density of seawater

These three processes are combined in one single equation by Buchanan 1988 to give:

$$\rho_{oil} = \left(Y\rho_w\right) + \rho_{ref}\left(1.0 - Y\right)\left(1.0 + CDensE.F_{evap}\right)\left(1.0 - CDensT\left(T - T_{ref}\right)\right)$$

Viscosity

The change in oil viscosity over time is related to three different processes; changing water temperature, evaporation, and emulsification.

The change in density due to changing temperature can be expressed using Andrade's correlations as:

$$\mu_{oil} = \mu_{ref} e^{CT\left(\frac{1}{T} - \frac{1}{T_{ref}}\right)}$$

where: μ_{oil} is viscosity of oil for given water temperature, T
μ_{ref} is original (reference) viscosity of oil at reference temperature T_{ref}
CT is an empirical constant (= 5000 (ref: ADIOS2))

The change in viscosity to evaporation can be expressed using Mackay's equation as:

$$\mu_{oil} = \mu_{ref}\, e^{CE \cdot F_{evap}}$$

where: F_{evap} is the fraction of oil evaporated from the slick
CE is an empirical constant (= 10.0 (ref: Reed 1998))

The change in viscosity due to emulsification can be expressed using Fingas' data from Table 2 as:

$$\mu_{oil} = \mu_{ref} . V_{Emul}$$

where: V_{emul} is the interpolated viscosity multiplier from Table 2.

These three processes are combined in one single equation to give:

$$\mu_{oil} = V_{emul} . \mu_{ref} . e^{CT\left(\frac{1}{T} - \frac{1}{T_{ref}}\right)} e^{CE.F_{evap}} .$$

IMPLEMENTATION

```
do P = 1 to stepT                          external timestep loop
    do IT =1 to stepIT                                  internal timestep loop

Spill Event
        if(ix(2) == pTS(n))then
            CALL InitialArea(RhoOil,DeltaRho,Phase1Time,AreaOil)
            ……
            CALL Distribute(AreaOil, x_diff,y_diff)
            CALL update_oil_particles(0, n, m, x_diff,y_diff)
        end if

Beaching
        if(nParLeft > 0 .and. VolumeOil > 0)then
            VolumeBeach    = VolumeOil * REAL(nParBeached)/REAL(nParLeft))
            VolumeBeached = VolumeBeached + VolumeBeach
            nParLeft = nParLeft - nParBeached
            nParbeached = 0
        end if
              …..
Spreading
        IF (Spreading) THEN
            CALL SpreadOptions(n,FirstAP,ElapsedTime,DeltaRho,                    &
                               AreaOil,VolumeOil,RhoOil,SprdCase,x_diff,y_diff)
            CALL update_oil_particles(SprdCase,n,m,x_diff,y_diff)
        END IF

Emulsion
        IF (Emulsification) THEN
            CALL Emulsify  (n,ElapsedTime,FirstAP,RhoOil,ViscOil,&
                                ResinOil,AsphOil,WaterContent,xviscemul)
        END IF

Evaporate
        IF (Evaporation) THEN
            CALL Evaporate (ElapsedTime,FirstAP,RhoOil,AreaOil,                   &
                            ResinOil,AsphOil,MassSpill,MassEvap,WaterContent,MassOil)
            CALL Dissolution(ElapsedTime,WaterContent,AreaOil,MassDiss)
        END IF

Dispersion
        IF (Dispersion) THEN
              CALL Disperse  (n,ElapsedTime,FirstAP,AreaOil,ViscOil,              &
                                RhoOil,par(:,pZ),MassOil,MassDisp)
        END IF

Density     CALL Density    (RhoOil,MassEvap,MassSpill,WaterContent,DeltaRho)

Viscosity   CALL Viscosity  (ViscOil,RhoOil,MassEvap,MassSpill, xviscemul)

            !update VolumeOil to reflect losses due oil weathering
            VolumeOil = VolumeSpill - VolumeBeached - VolumeEvap - VolumeDisp - VolumeDiss

Update      call update_particles()

    end do                                              internal timestep loop
end do                                  external timestep loop
```

REFERENCES

ADIOS2, 2000. ADIOS (Automated Data Inquiry for Oil Spills) version 2.0.1 online help manual". Hazardous Materials Response and Assessment Division, NOAA. Prepared for the U.S. Coast Guard Research and Development Centre.

Buchannan, I., Hurford, N. 1988. Methods for predicting the physical changes of oil spilled at sea. Oil and Chemical Pollution vol 4(4) pp311-328

CERC. 1984. Shore Protection Manual Vol.I. Coastal Engineering Research Centre, Dept of the Army, Waterways Experiment Station, USACE, Vicksburg, MS, USA

CONCAW. 1983. van Oudenhoven, J., Draper, V., et al. Characteristics of petroleum and its behavior at sea" CONCAWE Report No.8/83. Den Haag, November 1983.

Delvigne, G.A.L., Sweeney, C.E. 1988. Natural Dispersion of Oil. Oil Chem. Poll. 4:281-310

Fay, J. A. 1971. Physical processes in the spread of oil on a water surface. Proceedings of the Joint Conference on the Prevention and Control of Oil Spills. Washington DC. American Petroleum Institute p463-467

Fingas, M. 1997. The Evaporation of Oil Spills: Prediction of equations using distillation data. Arctic and Marine OilSpill Program Technical Seminar, Environment Canada. 1997 Vol1:20 pp1-20

Fingas, M. 2011. Models for Water-in-Oil Emulsion Formation in Chpt.10 of Oil Spill Science and Technology, 2011.Gulf Professional Publishing, UK. ISBN:978-1-85617-943-0

MOHID2.2003 downloaded from
http://maretec.mohid.com/PublicData/Products/Manuals/Mohid_Description.pdf

North, E. W., R. R. Hood, S.-Y. Chao, and L. P. Sanford. 2005. The influence of episodic events on transport of striped bass eggs to an estuarine nursery area. Estuaries 28(1): 106-121.

North, E. W., R. R. Hood, S.-Y. Chao, and L. P. Sanford. 2006a. Using a random displacement model to simulate turbulent particle motion in a baroclinic frontal zone: a new implementation scheme and model performance tests. Journal of Marine Systems 60: 365-380.

North, E. W., Z. Schlag, R. R. Hood, L. Zhong, M. Li, and T. Gross. 2006b. Modeling dispersal of *Crassostrea ariakensis* oyster larvae in Chesapeake Bay. Final Report to Maryland Department of Natural Resources, July 31, 2006. 55 p.

North, E. W., Z. Schlag, R. R. Hood, M. Li, L. Zhong, T. Gross, and V. S. Kennedy. 2008. Vertical swimming behavior influences the dispersal of simulated oyster larvae in a coupled particle-tracking and hydrodynamic model of Chesapeake Bay. Marine Ecology Progress Series 359: 99-115.

OILPOL. 2000. Rabeh, A.H., Lardner, R.W., Gunay, N. GulfSpill Version 2.0: a software package for oil spills in the Arabian Gulf" Environmental Modelling and Software 15 (2000) 425-442

Stiver, W., Mackay D. 1984. Evaporation rate of spills of hydrocarbons and petroleum mixtures. Environmental Science and Technology, 18(11): 834-840

APPENDIX I: OILTRANS_data input file

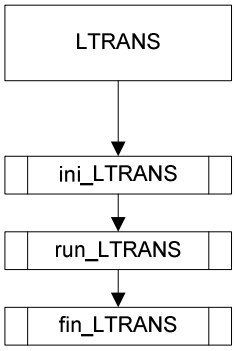
```
$numparticles                 **NUMBER OF PARTICLES**
  Numpar          = 100       Number of particles in simulation
$end
$timeparam                    **TIME PARAMETERS**
  Days            = 6.916667  Number of days to run the model
  Iprint          = 3600      Print interval for OILTRANS output (seconds)
  Dt              = 3600      External timestep (duration between ROMS/SWAN model predictions (s))
  Idt             = 120       Internal (particle tracking) timestep (s)
$end
$hydroparam                   **ROMS HYDRO MODEL PARAMETERS**
  Us              = 20        No. of Rho grid s-levels in ROMS model
  ws              = 21        No. of W grid s-levels in ROMS model
  tdim            = 1         No. of timesteps in each ROMS hydro archive file
  hc              = 0.2       Minimum depth – used in ROMS s-level transforms
  z0              = 0.0005    ROMS roughness parameter
  Vtransform      = 1         flag for vertical transform applied (1- WikiROMS Eq.1, 2 – WikiROMS Eq.2, 3 – Song/Haidvogel 1994 Eq.)
  readZeta        = .False.   if TRUE, read sea surface height, zeta, from ROMS netCDF archive
  constZeta       = 0         constant value for zeta if readZeta = .FALSE.
  readSalt        = .True.    if TRUE, read salinity from ROMS netCDF
  constSalt       = 0         constant value for salinity if readSalt = .FALSE.
  readTemp        = .True.    if TRUE, read temperature from ROMS netCDF
  constTemp       = 0         constant value for temperature if readTemp = .FALSE.
  readDens        = .False.   if TRUE, read density from ROMS netCDF
  constDens       = 0         constant value for density if readDens = .FALSE.
  readU           = .True.    if TRUE, read U velocity  from ROMS netCDF
  constU          = 0         constant value for U velocity  if readU = .FALSE.
  readV           = .True.    if TRUE, read V velocity  from ROMS netCDF
  constV          = 0         constant value for V velocity  if readV = .FALSE.
  readW           = .True.    if TRUE, read W velocity  from ROMS netCDF
  constW          = 0         constant value for W velocity  if readW = .FALSE.
  readAks         = .True.    if TRUE, read vertical salinity diffusion coefficient  from ROMS netCDF
  constAks        = 0         constant value for vertical salinity diffusion coefficient  if readAks = .FALSE.
$end
$turbparam                    **TURBULENCE PARAMETERS**
  HturbOn         = .False.   Horizontal turbulence on (TRUE) or off (FALSE)
  VTurbOn         = .False.   Vertical turbulence on (TRUE) or off (FALSE)
  ConstantHTurb   = 0         Constant value for horizontal turbulence if HTurbOn = TRUE
$end
$behavparam                   **BEHAVIOUR PARAMETERS** - NOT USED IN OILTRANS
  Behavior        = 0         Behaviour type (passive, near surface, near bottom, larval, oyster, etc)
  OpenOceanBoundary    = .True.       Allow particles to escape from model domain, ie stick to open ocean boundary (TRUE)
```

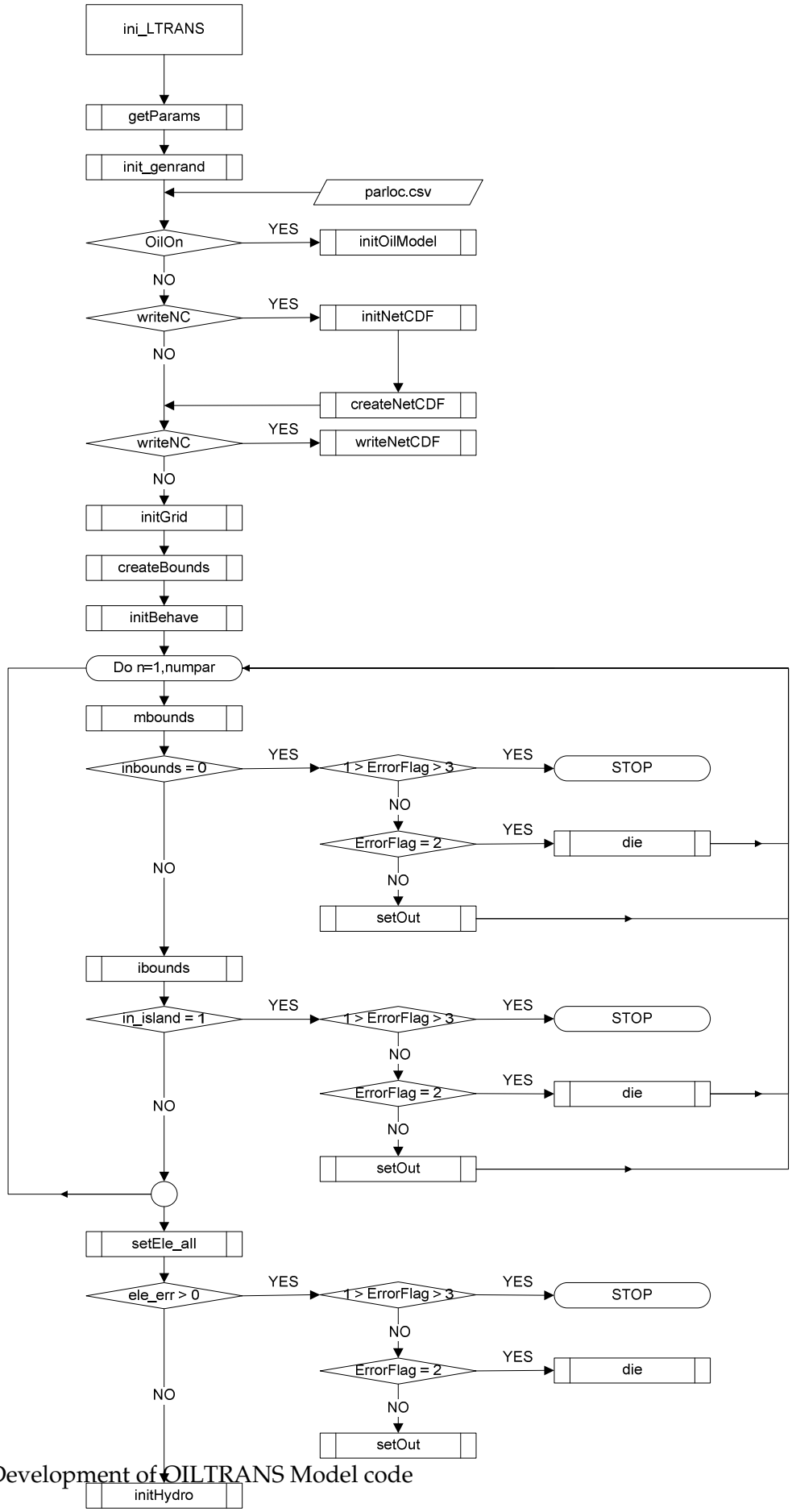
```
  mortality           = .False.     Allow particles to die (TRUE) or not (FALSE)
  deadage             = 691200      Age at which particles dies (stops moving) (s)
  pediage             = 345600      Age at which particle reaches max swimming speed and can settle (s)
  swimstart           = 0           Age that swimming (or sinking) begins (s)
  swimslow            = 0.005       Initial swimming speed (m/s)
  swimfast            = 0.043       Max swimming speed (m/s)
  Sgradient           = 1           Salinity gradient threshold that cues larval behaviour (psu/m)
  sink                = -0.0003     Sinking velocity (m/s)
  Hswimspeed          = 0.9         Tidal stream transport horizontal swimming speed (m/s)
  Swimdepth           = 2           Depth of swimming during flood tides (m) above bottom
$end
$dvmparam                           **DIURNAL VERTICAL MIGRATION PARAMETERS** - NOT USED IN OILTRANS
  twistart            = 4.801821    Time of twilight start(hr)
  twiend              = 19.19956    Time of twilight end (hr)
  daylength           = 14.39774    Length of a day (hr)
  Em                  = 1814.328    Irradiance at solar noon (uE. m^-2. s^-1)
  Kd                  = 1.07        Vertical attenuation coefficient
  thresh              = 0.0166      Light threshold that cues behaviour (uE. m^-2. s^-1)
$end
$settleparam                        **SETTLEMENT MODULE PARAMETERS** - NOT USED IN OILTRANS
  settlementon        = .False.     Settlement module on (TRUE) or off (FALSE)
  holesExist          = .False.     Are there holes in the habitat polygons
  minpolyid           = 101001      Lowest habitat polygon ID
  maxpolyid           = 101006      Highest habitat polygon ID
  minholeid           = 100101      Lowest hole number ID
  maxholeid           = 100501      Highest hole number ID
  pedges              = 36          Number of habitat polygon edge points (no of rows in habitat polygon file)
  hedges              = 12          Number of hole edge points (no of rows in holes polygon file)
$end
$convparam                          **CONVERSION PARAMETERS**
  PI                  = 3.141593    pi
  Earth_Radius        = 6378000     Equatorial radius of the earth
  SphericalProjection = .True.      Use spherical projection from ROMS
  latmin              = 53.5        Minimum latitude for spherical projection
  lonmin              = -10         Minimum longitude for spherical projection
$end
$romsgrid                           **ROMS netCDF model grid file & path
  NCgridfile          = '/home/marcel/PromINPUT/Connemara/GRID/connemara.nc'
$end
$romsoutput                                                                **ROMS netCDF model archived hydrodynamics file(s) and path
  prefix              = '/home/marcel/PromOUTPUT/Connemara/OILTRANS/connemara_his_'   NetCDF input file name prefix
  suffix              = '.nc'                                                NetCDF input file name suffix
  filenum             = 3363                                                 Number of first netCDF filename to use
  numdigits           = 4                                                    Number of digits in netCDF filename(s)
  startfile           = .False.                                              Is this the first file of a simulation (ie does it have initial timestep data)
$end
```

```
$parloc                                  **PARTICLE LOCATION INPUT FILE**
  parfile          = 'InitParLoc.csv'    Initial particle locations input file name (note file path must be included if file not in same folder as the model executable
$end
$habpolyloc                              **HABITAT POLYGON LOCATION INPUT FILE(S)** – NOT USED IN OILTRANS
  habitatfile      = 'habitat.csv'       Habitat locations input file name
  holefile         = 'holes.csv'         Habitat holes locations input file name
$end
$output                                  **OUTPUT RELATED VARIABLES** - NOT USED IN OILTRANS
  outpath           = './output/'        Folder structure to write output to
  NCOutFile        = 'output'            Name of netCDF output file(s)
  outpathGiven     = .False.             if TRUE then files are written to 'outpath'
  writeCSV         = .True.              If TRUE then write .csv output files
  writeNC          = .False.             If TRUE then write netCDF output files
  NCtime           = 0                   Time interval between creation of new netCDF output files
  SVN_Version      = ' '                 netCDF output file(s) metadata
  RunName          = ' '                 netCDF output file(s) metadata
  ExeDir           = ' '                 netCDF output file(s) metadata
  OutDir           = ' '                 netCDF output file(s) metadata
  RunBy            = ' '                 netCDF output file(s) metadata
  Institution      = ' '                 netCDF output file(s) metadata
  StartedOn        = ' '                 netCDF output file(s) metadata
$end
$other                                   **OTHER PARAMETERS
  seed             = 9                   Seed value for random number generator
  ErrorFlag        = 0                   What to do if an error is encountered (0-stop, 1-return particle to previous location, 2-kill particle, etc)
  BoundaryBLNs     = .False.             Create SURFER blanking file of boundaries (TRUE = yes, FALSE = no)
  SaltTempOn       = .False.             Calculate salinity and temperature at particle location
  TrackCollisions  = .False.             Write bottom hit and land hit files
  WriteHeaders     = .False.             Write .txt files with column headers
  WriteModelTiming = .False.             write .csv file with model timings
  ijbuff           = 4                   Number of extra elements to read on every side of the particle(s)
  OilOn            = .True.              Activate and use the OIL modules (TRUE = yes, FALSE = no)
$end
$oilparams                               **OIL MODULE PARAMETERS
  VolumeSpill      = 100                 Volume of oil spill (m3)
  SecSpill         = 0                   Starting time of oil spill wrt simulation start time (s)
  API              = 37                  Oil API gravity
  Oil_Dens         = 839                 Density of Oil (kg/m3)
  Oil_Dens_RefT    = 288                 Reference temperature at which density is calculated (oC)
  Watertemp        = 15.0                Water temperature (degC) – not used anymore in OILTRANS
  Dyn_Visc         = 0.0473              Dynamic viscosity of oil (cP)
  Kin_Visc         = 0.000056            Kinematic viscosity of oil (cSt)
  Dyn_Visc_RefT    = 273                 Reference temperature at which dynamic viscosity is calculated (oC)
  Kin_Visc_RefT    = 273                 Reference temperature at which kinematic viscosity is calculated (oC)
  Cut_Unit         = 'volume'            Units by which oil properties are determined (volume, weight)
```

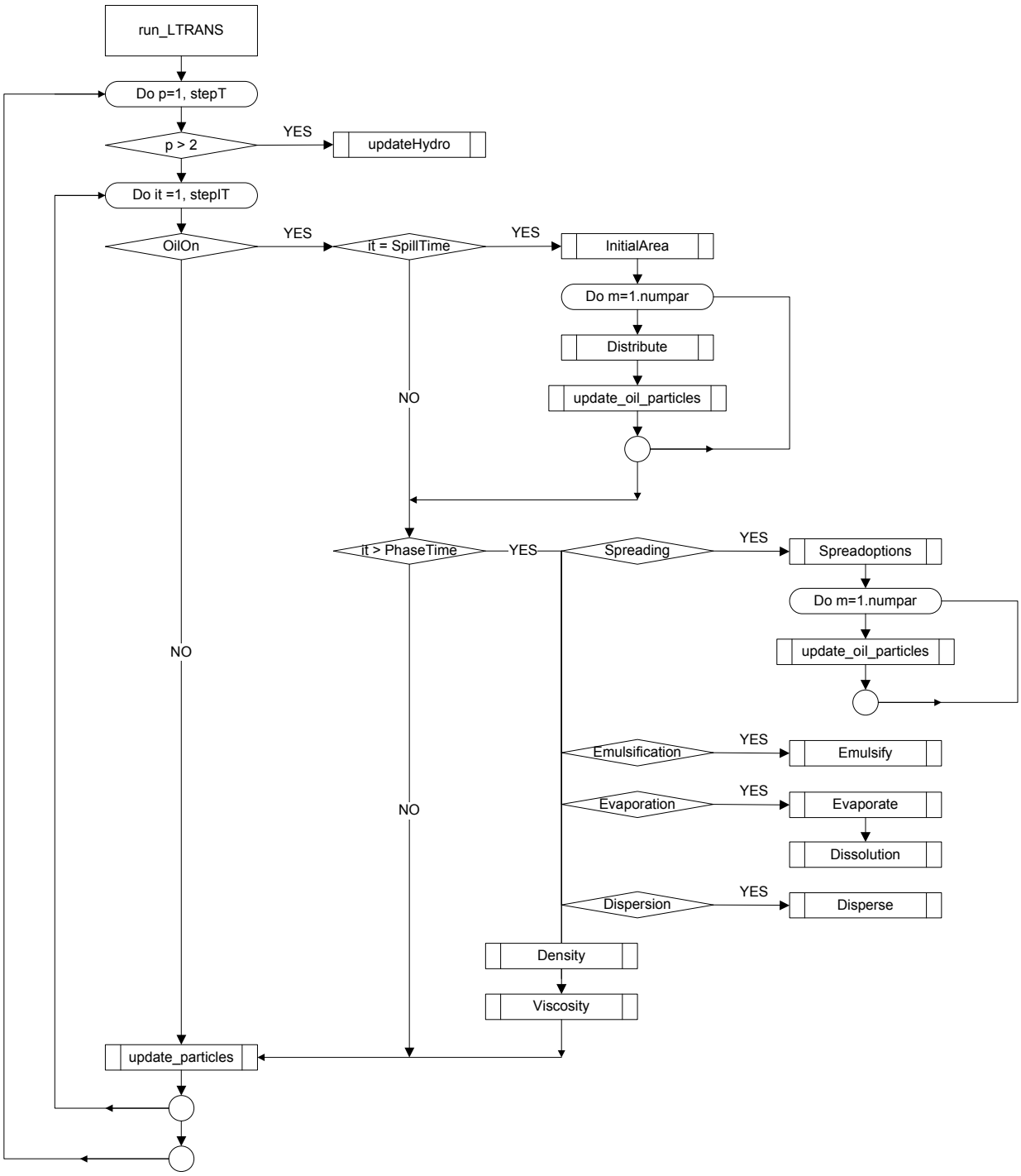
```
  Oil_Asph          = 0.02        Asphaltene content of oil
  Oil_Resin         = 20          Resin content of oil
  Oil_Sat           = 0.842       Saturates content of oil
  Cut_Temp          = 310,368,384,399,415,428,462,486,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0     Temperature of each oil cut at which properties are determined
$end
$oilprocs                       **OIL PROCESSES FOR MODELLING
  Spreading         = .False.     Turn on (TRUE) or off (FALSE) mechanical spreading of oil slick
  AreaOption        = 'ADIOS2'    Method for calculating initial area of oil slick (ADIOS2, MOHID, CONCAW, OILPOL)
  SprdOption        = 'ADIOS2'    Method for calculating mechanical spreading of oil slick ((ADIOS2, MOHID, CONCAW, OILPOL)
  Evaporation       = .False.     Turn on (TRUE) or off (FALSE) evaporation of oil slick
  EvapOption        = 'FINGAS'    Method ofor calculating evaporation from oil slick (FINGAS, MACKAY)
  Emulsification    = .False.     Turn on (TRUE) or off (FALSE) emuslification of oil slick
  Dispersion        = .False.     Turn on (TRUE) or off (FALSE) vertical dispersion of oil slick by wave action
  Langmuir          = .False.     Turn on (TRUE) or off (FALSE) movement of oil slick by langmuir circulation
  Stokes            = .False.     Turn on (TRUE) or off (FALSE) movement of oil slick by stokes drift
  Wind              = .False.     Turn on (TRUE) or off (FALSE) movement of oil slick by wind drift
$end
$windswaves                     **WINDS & WAVES MODULE PARAMETERS
  WindWaveModel     = .False.       Use SWAN model predictions
  swan_prefix       = '/swan_his_'  SWAN model output file prefix
  swan_filenum      = 1000          Number of first SWAN filename to use
  swan_suffix       = '.nc'         SWAN input file name suffix
  SigWaveHeight     = 10            Constant value for significant wave height if WindWaveModel = FALSE (recalculated in OILTRANS)
  SigWavePeriod     = 5             Constant value for significant wave period if WindWaveModel = FALSE
  SigWaveLength     = 5             Constant value for significant wave length if WindWaveModel = FALSE
  MeanWavePeriod    = 5             Constant value for mean wave period if WindWaveModel = FALSE
  UWind_10          = 2             Constant value for 10m U wind component if WindWaveModel = FALSE
  VWind_10          = 0             Constant value for 10m V wind component if WindWaveModel = FALSE
  PeakDirection     = 270           Constant value for peak wave direction if WindWaveModel = FALSE
  PeakWaveLength    = 10            Constant value for peak wave length if WindWaveModel = FALSE
  MixingDepth       = 5             Constant value breaking wave mixing depth if WindWaveModel = FALSE (recalculated in OILTRANS)
  Cd                = 0.001         Constant value for drag coefficient if WindWaveModel = FALSE
  Disper            = 0.001         Constant value for wave energy dispersion if WindWaveModel = FALSE
$end
```

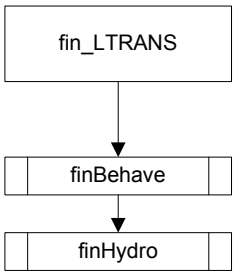
APPENDIX II: OILTRANS flow chart


LTRANS
ini_LTRANS
run_LTRANS
fin_LTRANS



Development of OILTRANS Model code


run_LTRANS
Do p=1, stepT
p > 2
YES
updateHydro
Do it =1, stepIT
OilOn
YES
it = SpillTime
YES
InitialArea
Do m=1.numpar
Distribute
update_oil_particles
NO
it > PhaseTime
YES
Spreading
YES
Spreadoptions
Do m=1.numpar
update_oil_particles
NO
Emulsification
YES
Emulsify
Evaporation
YES
Evaporate
Dissolution
NO
Dispersion
YES
Disperse
Density
Viscosity
update_particles


fin_LTRANS
finBehave
finHydro

APPENDIX III: OILTRANS CODE

Subroutine INITOIL

```
!****************************************
!*     Subroutine InitOil              *
!****************************************
SUBROUTINE InitOilModel(RhoOil,OilDensity,OilDensity_RefT,DeltaRho,ViscOil)

USE PARAM_MOD,    ONLY:          Cut_Unit,Cut_Temp,Oil_Dens,WaterTemp, Oil_Dens_RefT,API,Dyn_Visc,Dyn_Visc_RefT,        &
                  Kin_Visc,Kin_Visc_RefT

IMPLICIT NONE

!I/O variables
DOUBLE PRECISION, INTENT(OUT):: RhoOil,OilDensity,OilDensity_RefT,DeltaRho,ViscOil

!Local variables
DOUBLE PRECISION:: SGOil15                                  !Specific gravity of oil at 15degC
DOUBLE PRECISION:: RhoOil15                                 !Density of oil at 15degC (kg/m3)
INTEGER:: i, n, NumCuts                                     !counters
LOGICAL:: novalue                                   !logic controllers
DOUBLE PRECISION:: LowPresTemps(5)                          !array for holding converted 40mmHG pressure temperatures

!*****************************
!*   INTIALIsE VARIABLES   *
!*****************************
! Equates to     200.0, 225.0, 250.0, 275.0, 300.0 degC at 40mmHg
LowPresTemps = (/307.8, 337.7, 365.8, 394.9, 424.0/) !(K)
! see BPO Crude Oil Analysis Data Bank User's Guide Methods
! Taken from standard pressure-temperature nomograph

novalue = .FALSE.

!*********************************
!*   INTIALISE OIL PROPERTIES   *
!*********************************

!Determine no. of cuts for volume distillation only
!All other options use the ADIOS correlation eqn for
!pseudocomponent evaporation
SELECT CASE (trim(adjustl(Cut_Unit)))
      CASE ("-----")
            NumCuts = 0

      CASE ("weight")
            NumCuts = 0

      CASE ("volume")
            n = 1
            NumCuts = 0
            !determine number of cuts
            DO WHILE (.NOT.novalue .AND. n < 16)                !n = 15 [max number of cuts (10@760mmHg & 5@40mmHg)]
                  IF (Cut_Temp(n) == 0.0) THEN
                        novalue = .TRUE.
                  ELSE
                        NumCuts = n
                        n = n + 1
                  END IF
            END DO
            !determine if cuts have been made at reduced pressure
            i = 1
```

```
            DO n = 1, NumCuts-1
                IF (Cut_Temp(n) > Cut_Temp(n+1)) THEN                    !if temperature steps down => reduced pressure distillation
                    Cut_Temp(n+1) = LowPresTemps(i)   + 273.15           !assign temp correction based on P-T nomograph
                    i = i + 1
                END IF
            END DO

        CASE DEFAULT
            WRITE(*,*)'Error: Cut_Unit not defined'
            WRITE(*,*)'*** FATAL ERROR - STOP ***'
            STOP

END SELECT

!Determine initial oil density
IF (Oil_Dens > 0.0) THEN
    RhoOil = Oil_Dens * (1.0 - CDensT * ((WaterTemp+273.15) - Oil_Dens_RefT))    !actual density of spilled oil at ocean temperature (kg/m3)
    OilDensity = Oil_Dens
    OilDensity_RefT = Oil_Dens_RefT
ELSEIF (API > 0.0) THEN
    SGOil15 = 141.5 / (131.5 + API)                                  !specific gravity of spilled oil at reference temp (15.5degC)    - from API standards
    RhoOil15 = SGOil15 * RhoFWater15                                 !density of spilled oil at reference temp (15.5degC)
    RhoOil = RhoOil15 * (1.0 - CDensT * (WaterTemp - WaterTemp15))   !actual density of spilled oil at ocean temperature (kg/m3)
    OilDensity = RhoOil15                                            !Reference oil density for subroutine DENSITY
    OilDensity_RefT = WaterTemp15 + 273.15                           !reference oil temperature for subroutine DENSITY
ELSE
    WRITE(*,*)'No value DENSITY associated with this oil'
    WRITE(*,*)'No modelling can be done'
    WRITE(*,*)'*** FATAL ERROR - STOP ***'
    STOP
END IF

DeltaRho  = (RhoWater - RhoOil) / RhoWater                           !relative density difference between water and oil densities

!Determine initial oil dynamic viscosity
IF (Dyn_Visc > 0.0) THEN
     ViscOil = 1000.0 * Dyn_Visc * exp(ViscCt * ((1.0/(WaterTemp+273.15)) - (1.0/Dyn_Visc_RefT)))    !Dynamic (kg/ms -> cP)
ELSE
    ViscOil = Kin_Visc * exp(ViscCt * 1.0/(WaterTemp+273.15)) - (1.0/Kin_Visc_RefT)))    !Kinematic (m2/s -> cSt)
    write(*,*)'kin:',viscoil
    ViscOil = 1000.0 * ViscOil * RhoOil                              !Dynamic (cP)
END IF

write(*,*)viscoil

RETURN

END SUBROUTINE InitOilModel
```

Subroutine INITIALAREA

```
!*******************************
!*    Subroutine InitialArea *
!*******************************
SUBROUTINE InitialArea(RhoOil,DeltaRho,Phase1Time,OilArea)
!Calculates the time to the end of the gravity-inertial spreading phase
!(which we don't model)
!and calculates the area of the spreaded oil at the end of that phase.
USE PARAM_MOD, ONLY: AreaOption, pi, VolumeSpill

IMPLICIT NONE
```

```
!I/O variables
DOUBLE PRECISION, INTENT(IN) :: RhoOil, DeltaRho
INTEGER, INTENT(OUT):: Phase1Time                    !Computed time to gravity-inertial spreading phase (sec)
DOUBLE PRECISION, INTENT(OUT) :: OilArea

!Local variables
DOUBLE PRECISION:: Area1
DOUBLE PRECISION:: Area2
DOUBLE PRECISION:: Time

!When oil is denser than surrounding water, oil will sink => no spreading
IF (RhoOil > RhoWater) THEN
      !Oil is denser than water => no surface spreading
      RETURN
END IF

SELECT CASE (F_UpCase(trim(adjustl(AreaOption))))

      CASE ("MOHID2")
            !MOHID Description, 2003
            !downloaded from
            !http://maretec.mohid.com/PublicData/Products/Manuals/Mohid_Description.pdf
            OilArea     = pi * (1.45**4.0 / 1.14**2.0) * 0.25 * (VolumeSpill**5.0 * Gravity * DeltaRho / (KinViscWater**2.0))**(1.0/6.0)
            Phase1Time = nint((0.725/0.570)**4.0 * (VolumeSpill / (KinViscWater * Gravity * DeltaRho))**(1.0/3.0))

      CASE ("ADIOS2")
            !NOAA, 2000
            !"ADIOS (Automated Data Inquiry for Oil Spills)
            !version 2.0.1 online help manual"
            !Hazardous Materials Response and Assessment Division,NOAA.
            !Prepared for the U.S. Coast Guard Research and Development Center,
            OilArea     = pi * (1.21**4.0 / 1.53**2.0) * (VolumeSpill**5.0 * Gravity * DeltaRho  / (KinViscWater**2.0))**(1.0/6.0)
            Phase1Time = nint((1.21/1.53)**4.0 *  (VolumeSpill / (KinViscWater * Gravity * DeltaRho))**(1.0/3.0))

      CASE ("CONCAW")
            !van Oudenhoven, J., Draper, V., et al 1983
            !"Characteristics of petroleum and its behavior at sea"
            !CONCAWE Report No.8/83. Den Haag, November 1983.
            Time = 0.00                                          !seconds (ie: almost instantaneous)
            Area1 = 0.0                                          !gravity-inertia
            Area2 = 0.0                                          !gravity-viscous
            DO WHILE (Area1 < Area2)                                   !until gravity-viscous regime reached
                  Area1 = pi * (1.14**2.0) *  (DeltaRho * Gravity * VolumeSpill)**0.5 * Time
                  Area2 = pi * (0.98**2.0) *  ((DeltaRho * Gravity * (VolumeSpill**2.0)) / (KinViscWater**0.5))**(1.0/3.0) * (Time**0.5)
                  Time = Time + 10.0
            END DO
            OilArea = Area2                                      !Area at start of gravity-viscous regime
            Phase1Time = nint(Time)                                    !Time since spill to start of gravity-viscous regime

      CASE ("OILPOL")    !GULFSPILL
            !Rabeh, A.H., Lardner, R.W., Gunay, N. 2000
            !"GulfSpill Version 2.0: a software package for oil spills in the Arabian Gulf"
            !Environmental Modelling and Software 15 (2000) 425-442
            OilArea = pi * (2.81 * sqrt(VolumeSpill))**2.0
            Phase1Time = nint(0.00)

      CASE DEFAULT
            WRITE(*,*)'      Case not encoded'
            WRITE(*,*)'No INITIAL SPILL AREA calculated'
            WRITE(*,*)'****** PROGRAM TERMINATING ******'
            OilArea = 0.0
```

Development of OILTRANS Model code

```
            Phase1Time = 0.0

END SELECT

RETURN

END SUBROUTINE InitialArea
```

Subroutine DISTRIBUTE

```
!*********************************
!*    Subroutine Distribute   *
!*********************************
SUBROUTINE Distribute(OilArea, XDiff, YDiff)
!Initially randomly distribute particles using a normal distribution
!throughout the theoretical Fay area of spill
USE PARAM_MOD, ONLY: pi
IMPLICIT NONE

!I/O variables
DOUBLE PRECISION, INTENT(IN):: OilArea
DOUBLE PRECISION, INTENT(OUT):: XDiff
DOUBLE PRECISION, INTENT(OUT):: YDiff

!Local variables
DOUBLE PRECISION:: ran                  !holder for random number
DOUBLE PRECISION:: ang                  !randomly generated angle (0 < ang < 2pi)
DOUBLE PRECISION:: dist                 !randomly generated distance (0 < dist < Radius)
DOUBLE PRECISION:: lenR                 !(m) Initial Radius of spilled oil at end of inertial spreading

lenR = sqrt(OilArea / pi)

CALL random_number(ran)
ang = ran * pi * 2.0

CALL random_number(ran)
dist = ran * lenR

XDiff = dist * cos(ang)
YDiff = dist * sin(ang)

END SUBROUTINE Distribute
```

Subroutine update_oil_particles

```
!*****************************************
!*    Subroutine update_oil_particles*
!*****************************************
SUBROUTINE update_oil_particles(s, n, m, param_1, param_2)
  USE PARAM_MOD,              ONLY: idt,numpar,pi

  IMPLICIT NONE

!I/O variables
INTEGER, INTENT(IN) :: s,n,m
DOUBLE PRECISION, INTENT(IN):: param_1, param_2

!local variables
```

```
DOUBLE PRECISION :: ran1,ran2,Ud,Vd

SELECT CASE (s)

    !Initial distribution after release
    CASE (0)
        par(m,pnX)  = par(m,pX) + param_1
        par(m,pnY)  = par(m,pY) + param_2

    !MOHID Diffusion
    CASE(1)
        CALL random_number(ran1)
        CALL random_number(ran2)

        Ud = ran1 * cos(2.0 * pi * ran2) * param_1                  !param_1 = DiffVelocity
        Vd = ran1 * sin(2.0 * pi * ran2) * param_2                  !param_2 = DiffVelocity

        par(m,pnX) = par(m,pX) + (Ud * idt)
        par(m,pnY) = par(m,pY) + (Vd * idt)

    !ADIOS spreading radius (only using CoefR: param_1)
    CASE(2)
        CALL random_number(ran1)
        CALL random_number(ran2)

        Ud = ran1 * cos(2.0 * pi * ran2) * param_1                  !param_1 = DiffVelocity
        Vd = ran1 * sin(2.0 * pi * ran2) * param_1                  !param_1 = DiffVelocity

        par(m,pnX) = par(m,pX) + (Ud * idt)
        par(m,pnY) = par(m,pY) + (Vd * idt)

    !CONCAW circular spreading radius
    CASE(3)
        par(m,pnX) = ((par(m,pX) - par((((numpar/nts)*(n-1))+1),pX)) * param_1) + par((((numpar/nts)*(n-1))+1),pX)
        par(m,pnY) = ((par(m,pY) - par((((numpar/nts)*(n-1))+1),pY)) * param_1) + par((((numpar/nts)*(n-1))+1),pY)

    !OILPOL ellipsical spreading radius
    CASE(4)
        par(m,pnX) = ((par(m,pX) - par((((numpar/nts)*(n-1))+1),pX)) * param_1) + par((((numpar/nts)*(n-1))+1),pX)
        par(m,pnY) = ((par(m,pY) - par((((numpar/nts)*(n-1))+1),pY)) * param_2) + par((((numpar/nts)*(n-1))+1),pY)

    CASE DEFAULT !Initial distribution of particels
         par(m,pnX)  = par(m,pX)
         par(m,pnY)  = par(m,pY)

END SELECT

par(m,pX)  = par(m,pnX)
par(m,pY)  = par(m,pnY)
par(m,pZ)  = par(m,pnZ)

RETURN
END SUBROUTINE update_oil_particles
```

Subroutine SPREADOPTIONS

```
!****************************************
!*     Subroutine SpreadOptions     *
!****************************************
SUBROUTINE SpreadOptions(n,FirstAP,ElapsedTime,DeltaRho,OilArea,VolumeOil, ,SprdCase,CoefR,CoefQ)
```

```
USE PARAM_MOD, ONLY: PI,SprdOption,idt,numpar,Uwind_10,Vwind_10

IMPLICIT NONE

INTEGER, INTENT(IN) :: n, ElapsedTime
LOGICAL, INTENT(IN) :: FirstAP(n)
DOUBLE PRECISION, INTENT(IN) :: DeltaRho, VolumeOil,RhoOil

DOUBLE PRECISION, INTENT(INOUT) :: OilArea

INTEGER, INTENT(OUT) :: SprdCase
DOUBLE PRECISION, INTENT(OUT) :: CoefR, CoefQ

DOUBLE PRECISION:: WindSpeed                    !MOHID
DOUBLE PRECISION:: DiffCoef                     !MOHID
DOUBLE PRECISION:: DiffVelocity                 !MOHID
DOUBLE PRECISION:: beta                         !MOHID
DOUBLE PRECISION:: beta_old                     !MOHID
DOUBLE PRECISION:: aux1                         !MOHID
DOUBLE PRECISION:: aux2                         !MOHID
DOUBLE PRECISION:: Area                         !CONCAWE & ADIOS
DOUBLE PRECISION:: Area2                        !CONCAWE & ADIOS
DOUBLE PRECISION:: Area3                        !CONCAWE & ADIOS
DOUBLE PRECISION:: MaxArea                      !CONCAWE & ADIOS
DOUBLE PRECISION:: DeltaR                       !CONCAWE & ADIOS
DOUBLE PRECISION:: DeltaQ                       !OILPOL
DOUBLE PRECISION:: lenR                         !ADIOS
DOUBLE PRECISION:: lenQ                         !OILPOL
DOUBLE PRECISION:: lenR1                        !OILPOL
DOUBLE PRECISION:: lenQ1                        !OILPOL
DOUBLE PRECISION:: Dfay                         !ADIOS2

!save values on exit
SAVE:: beta_old,lenR1,lenQ1

SELECT CASE (F_UpCase(SprdOption))

    !FAY based method
    CASE ("MOHID2")
        !MOHID Description, 2003
        !downloaded from
        !http://maretec.mohid.com/PublicData/Products/Manuals/Mohid_Description.pdf
        !& with reference to source code.
        SprdCase = 1
            beta = pi * ((1.45**2)/4.0) * (DeltaRho * Gravity * (VolumeOil**2) / sqrt(KinViscWater))**(1.0/3.0)

        IF (FirstAP(n)) THEN
                beta_old = beta
            END IF

            aux1  = beta_old
            aux2  = OilArea/aux1
        beta_old = beta
        Area  = beta * sqrt((aux2)**2 + IDT)

        if(Area < OilArea)then
            Area = OilArea
        end if

        DiffCoef = (pi * (0.725**2.0))/16.0 * ((DeltaRho * Gravity * (VolumeOil**2.0))          &
                                / sqrt(KinViscWater))**(1.0/3.0)        &
                                * (1.0/ sqrt (REAL(ElapsedTime,kind(1))))
```

```
        DiffVelocity = sqrt ((2.0 * DiffCoef) / IDT)
        CoefR = DiffVelocity
        CoefQ = DiffVelocity

    !Fay based method
    CASE ("ADIOS2")
        !NOAA, 2000
        !"ADIOS (Automated Data Inquiry for Oil Spills)
        !version 2.0.1 online help manual"
        !Hazardous Materials Response and Assessment Division,NOAA.
        SprdCase = 2
        Area = pi * (1.21**2.0) * (VolumeOil**2.0 * Gravity * DeltaRho  * (ElapsedTime**(3.0/2.0))              &
                                / sqrt(KinViscWater))**(1.0/3.0)

        if(Area < OilArea)then
            Area = OilArea
        end if

        !Diffusion ceofficient
        DiffCoef = ((1.21**2.0)/16.0) * (((VolumeOil**2.0 * Gravity * DeltaRho)                          &
                                / sqrt(KinViscWater))**(1.0/3.0))              &
                                * (1.0/sqrt (REAL(ElapsedTime,kind(1))))

        DiffCoef = DiffCoef + (0.033 * ((ElapsedTime)**0.16))
        DiffVelocity = sqrt((2.0 * DiffCoef) / IDT)

        lenR = sqrt(Area/pi)                                !NewRadius of updated Area of Oil
        DeltaR     = lenR - sqrt(OilArea/pi)                            !NewRadius - OldRadius
        CoefR      = DeltaR / lenR                      !DeltaR / NewRadius
!       CoefR      = lenR / sqrt(OilArea/pi)                            !fractional Radius increase
!       CoefQ      = 0.0
        CoefR = DiffVelocity
        OilArea = Area                                      !Update area of oil


    !FAY based method
    CASE ("CONCAW")
        !van Oudenhoven, J., Draper, V., et al 1983
        !"Characteristics of petroleum and its behavior at sea"
        !CONCAWE Report No.8/83. Den Haag, November 1983.
        SprdCase = 3
        Area2 = pi * (0.98**2.0) * ((DeltaRho * Gravity * (VolumeOil**2.0))                          &
                        /(KinViscWater**0.5))**(1.0/3.0) * (ElapsedTime**0.5)

        Area3 = pi * (1.60**2.0) * (((SpreadCoeff * 10**-3.0)**2.0)                          &
                        / ((RhoWater**2.0) * KinViscWater))**0.5 *(ElapsedTime**1.5)
        MaxArea = 10**5.0 * (VolumeOil**0.75)
        Area = Area2

        IF (Area2 < Area3) THEN
            Area = Area3
            IF (Area3 > MaxArea) THEN
                Area = MaxArea
            END IF
        END IF

        if(Area < OilArea)then
            Area = OilArea
        end if

        lenR    = sqrt(Area/pi)
```

```
!           DeltaR  = lenR - sqrt(OilArea/pi)                    !NewRadius - OldRadius
!           CoefR     = DeltaR / lenR                          !DeltaR / NewRadius
            CoefR     = lenR / sqrt(OilArea/pi)                      !fractional Radius increase
            CoefQ     = 0.0
            OilArea = Area                                  !Updated area of oil

      !Lehr based method
      CASE ("OILPOL")    !GULFSPILL
            !OILPOL_2 solution - see OILPOL2.xls
            !Chao, X., Shankar, J., Wang, S. 2003
            !"Development and application of oil spill model for Singapore coastal waters"
            !Journal of Hydraulic Engineering 129:7 (2003) 495-503
            SprdCase = 4
            WindSpeed = sqrt((Uwind_10**2.0) + (Vwind_10**2.0))
            Area = 2270.0 * (DeltaRho * (RhoWater/RhoOil))**(2.0/3.0) * (VolumeOil * m32bbl)**(2.0/3.0)          &
                        * (ElapsedTime * sec2min)**(1.0/2.0)                        &
                        + (40.0 * (DeltaRho * (RhoWater/RhoOil))**(1.0/3.0)         &
                        * (ElapsedTime * sec2min) * (WindSpeed * ms2kts)**(4.0/3.0))

            if(Area < OilArea)then
                  Area = OilArea
            end if

            if(FirstAP(n))then
                  lenQ1 = lenQ
                  lenR1 = lenR
            end if

            lenQ = 53.76 * (DeltaRho * (RhoWater/RhoOil))**(1.0/3.0) *(VolumeOil * m32bbl)**(1.0/3.0)                &
                        * (ElapsedTime * sec2min)**(1.0/4.0)
            lenR = lenQ + 0.95 * (WindSpeed * ms2kts)**(4.0/3.0)*(ElapsedTime * sec2min)**(3.0/4.0)

            if(FirstAP(n))then
                  lenQ1 = lenQ
                  lenR1 = lenR
            end if

            CoefR = lenR / lenR1
            CoefQ = lenQ / lenQ1
            lenR1 = lenR
            lenQ1 = lenQ

      CASE DEFAULT
            WRITE(*,*)'Case not encoded'
            WRITE(*,*)'No SPREADING processes modelled'
      END SELECT

RETURN

END SUBROUTINE SpreadOptions
```

Subroutine EVAPORATE

```
!*****************************
!*    Subroutine Evaporate  *
!*****************************
SUBROUTINE Evaporate(ElapsedTime,FirstAP,RhoOil,AreaOil,ResinOil,AsphOil, MassSpill,MassEvap,WaterContent,MassOil)

USE PARAM_MOD, ONLY:      WaterTemp,Uwind_10,Vwind_10,EvapOption,idt,VolumeSpill,Oil_Resin, Oil_Asph
IMPLICIT NONE
```

```
!I/O variables
INTEGER, INTENT(IN):: ElapsedTime
LOGICAL, INTENT(IN):: FirstAP
double precision, intent(in):: MassSpill, RhoOil,AreaOil, WaterContent,MassOil
double precision, intent(inout):: resinoil, asphoil
double precision, intent(out):: MassEvap

!Local variables
DOUBLE PRECISION:: PercentEvap                          !percent of oil evaporated in timestep
DOUBLE PRECISION:: CumPercentEvap                       !cumulative percentage of oil evaporated to date
DOUBLE PRECISION:: PercentDist                          !percent of oil mass distilled at 180degC (FINGAS only)
INTEGER:: n                                     !counter
DOUBLE PRECISION:: Mol(16)                              !molecular weight of pseudocomponent
DOUBLE PRECISION:: MolFrac(16)                          !molar fraction of pseudocomponent
DOUBLE PRECISION:: VolFrac(16)                          !volume fraction of pc
DOUBLE PRECISION:: AvgMW                                !average molecular weight
DOUBLE PRECISION:: Ke                                   !mass transfer coefficient (m/s)
DOUBLE PRECISION:: VEvap(16)                            !volume of each pc evaporated
DOUBLE PRECISION:: dTdFe                                !rate of change of temperature versus fraction evaporated
DOUBLE PRECISION:: InitBP                               !initial boiling point
INTEGER:: nPC                                   !number of pseudocomponent
DOUBLE PRECISION :: WindSpeed
double precision :: VolEvap

!save values on exit
SAVE:: CumPercentEvap, dTdFe, InitBP, nPC, PercentDist

IF (FirstAP) THEN
      CALL InitialEvap(dTdFe, InitBP, nPC)
      PercentEvap = 0.0
      CumPercentEvap = 0.0
      MassEvap = 0.0
      PercentDist = 20.0                                !Alberta Sweet Mixed Blend (Fingas book p 219)
END IF

SELECT CASE (F_UpCase(EvapOption))

      CASE ("FINGAS")
            !Fingas, M. 1997
            !"The Evaporation of Oil Spills:
            !Prediction of equations using distillation data"
            !Arctic and Marine OilSpill Program Technical Seminar,
            !Environment Canada. 1997 Vol1:20 pp1-20

            !Assume Logarithmic (vast majority of oil types
            PercentEvap = ((0.165 * PercentDist) + (0.045 * (WaterTemp-15.0))) * log(REAL(ElapsedTime,KIND(1)) / 60.0)
            !!SquareRoot only applies to a few refined products -
            !!eqn below (may include later)
            !CumPercentEvap = ((0.0254 * PercentDist) + (0.01 * (WaterTemp- 5.0)))&
            !                       * sqrt(ElapsedTime / 60.0)

            MassEvap = MassSpill * (PercentEvap / 100.0)
            ResinOil = ResinOil / (1.0 - (MassEvap/MassSpill))              !Increase in resin % (Assuming no evaporation)
            AsphOil = AsphOil / (1.0 - (MassEvap/MassSpill))                !Increase in asphaltene % (Assuming no evaporation)

!     CASE ("PSEUDO")
!           !ADIOS2
!           DO n = 1, nPC
!                 Mol(n) = Vol(n) / Vbar(n)                 !moles in volume of pseudo component
!           end do
!
!           do n = 1, nPC
```

```
!                 IF (Vol(n) == 0.0)THEN
!                     MolFrac(n) = 0.0
!                     VolFrac(n) = 0.0
!                 ELSE
!                     MolFrac(n) = Mol(n) / sum(Mol)
!                     VolFrac(n) = Vol(n) / sum(Vol)
!                     AvgMW = AvgMW + MolFrac(n)*MW(n)
!                 END IF
!             END DO
!
!             Ke = 0.0048 * WindSpeed**(7.0/9.0) * 1.3676 * (sqrt(0.018/AvgMW))**(2.0/3.0)) * (lenR*2.0)**(-1.0/9.0)
!
!             DO n = 1, nPC
!                 VEvap(n) = (Ke * VolumeOil * VapP(n) * Vbar(n) * VolFrac(n)) / &
!                             (R * Slickthickness * (WaterTemp+273.15)) * idt
!                 IF (Vol(n) - VEvap(n) > 0) THEN
!                     Vol(n) = Vol(n) - VEvap(n)
!                 ELSE
!                     Vol(n) = 0.0
!                 END IF
!             END DO
!             VolumeEvap = VolumeEvap + sum(VEvap)
!             MassEvap = VolumeEvap * RhoOil

        CASE ("MACKAY")
            !Stiver W., Mackay, D. 1984
            !"Evaporation rate of spills of hydrocarbons and petroleum mixtures"
            !Environmental Science and Technology, 1984. vol 18, pp 834-480

            WindSpeed = sqrt((Uwind_10**2.0) + (Vwind_10**2.0))
            Ke = 1.5E-3 * WindSpeed**0.78
            PercentEvap = ((Ke * AreaOil * idt) / VolumeSpill) * exp(6.3 - ((10.3 *(InitBP + (dTdFe*CumPercentEvap))) &
                          / (WaterTemp + 273.15)))

            CumPercentEvap = CumPercentEvap + PercentEvap                   !Cumulative percentage evaporated
            VolEvap  = VolumeSpill * CumPercentEvap                 !Total volume evaporated to date
            MassEvap = VolEvap * RhoOil                     !Total mass evaporated to date
            ResinOil = Oil_Resin / (1.0 - (MassEvap/MassSpill))             !Increase in resin % (Assuming no evaporation of resins)
            AsphOil = Oil_Asph / (1.0 - (MassEvap/MassSpill))               !Increase in asphaltene % (Assuming no evaporation of asphaltenes)

        CASE DEFAULT
!           VolumeEvap = 0.0
            MassEvap = 0.0
            WRITE(*,*)'Case not encoded'
            WRITE(*,*)'No EVAPORATION processes modelled'

    END SELECT

    END SUBROUTINE Evaporate
```

Subroutine EMULSIFY

```
!*****************************
!*    Subroutine Emulsify   *
!*****************************
SUBROUTINE Emulsify(p,ElapsedTime,FirstAP,RhoOil,ViscOil,ResinOil,AsphOil, WaterContent, xviscemul)
!Fingas,M.,2011, "Models for Water-in-Oil Emulsion Formation" in
!Chpt. 10 of Oil Spill Science and Technology, 2011.
!Gulf Professional Publishing, UK. ISBN:978-1-85617-943-0
USE PARAM_MOD, ONLY: Oil_Sat,idt,SigWaveHeight
```

```
IMPLICIT NONE

!I/O variables
INTEGER, INTENT(IN):: ElapsedTime,p
double precision, intent(in):: RhoOil, ViscOil, ResinOil,AsphOil
double precision, intent(out):: WaterContent, xviscemul
logical, intent(in):: FirstAP(p)

!Local variables
DOUBLE PRECISION:: RelRhoOil                    !Relative density (decimal)
DOUBLE PRECISION:: D_t                          !transformed density
DOUBLE PRECISION:: V_t                          !transformed viscosity
DOUBLE PRECISION:: S_t                          !transformed saturates
DOUBLE PRECISION:: R_t                          !transformed resins
DOUBLE PRECISION:: A_t                          !transformed asphaltenes
DOUBLE PRECISION:: AR                           !asphaltene/resin ratio
DOUBLE PRECISION:: AR_t                         !transformed asphaltene/resin ratio
DOUBLE PRECISION:: StabilityC                   !StabilityC index
DOUBLE PRECISION:: Class(4,8)                   !Array to hold Fingas empricial values
DOUBLE PRECISION:: FormTime                     !Time for emulsion state to form (min)
DOUBLE PRECISION:: StartTime                    !Start time from which to begin timing emulsion formation, etc (sec)
DOUBLE PRECISION:: FingasDay                    !Seconds from StartTime to end of one day (including formation time)
DOUBLE PRECISION:: FingasWeek                   !as above to end of one week
DOUBLE PRECISION:: FingasYear                   !as above to end of one year
INTEGER:: ClassIndex                            !Class Index (1 - 4)
INTEGER:: StartClass                            !class index and integer
INTEGER::n                                      !class index and integer
double precision, parameter:: eps = 1e-6

!save values on exit
SAVE:: ClassIndex, FormTime, StartClass, StartTime, FingasDay, FingasWeek, FingasYear

IF (FirstAP(p)) THEN
!Fingas,M.,2011, "Models for Water-in-Oil Emulsion Formation" in
!Chpt. 10 of Oil Spill Science and Technology, 2011.
!Gulf Professional Publishing, UK. ISBN:978-1-85617-943-0
                !  | U    E     M     S   |
      Class = reshape((/    0.06,  0.42,  0.64,  0.76, &          ! DayWaterContent <- Figure 10.4
                  0.06,  0.37,  0.32,  0.76, &                    ! WeekWaterContent <- Figure 10.4
                  0.06,  0.37,  0.20,  0.68, &                    ! YearWaterContent <- Figure 10.4
                  1.0,   1.9,   7.2, 405.0, &                     ! DayViscosityIncrease <-Table 10.4
                  1.0,   1.9,  11.0,1054.0, &                     ! WeekViscosityIncrease <-Table 10.4
                  1.0,   2.1,  32.0, 991.0, &                     ! YearViscosityIncrease <-Table 10.4
                  0,  30.8,  47.0,  27.1, &                       ! FormTimeParamA <- Table 10.5
                  0, 18300, 49100,  7520 /), &                    ! FormTimeParamB <- Table 10.5
                       shape(Class))

      WaterContent = 0.0
      xViscEmul = 1.0
      ClassIndex = 0
      StartClass = 0
      StartTime = 0.0
END IF

IF(ClassIndex < 4 ) THEN                                          !check until stable emulsion forms

      !Fingas,M.,2011, "Models for Water-in-Oil Emulsion Formation" in
      !Chpt. 10 of Oil Spill Science and Technology, 2011.
      !Gulf Professional Publishing, UK. ISBN:978-1-85617-943-0

      !Density transform (ranges checked)
```

```
RelRhoOil = RhoOil / 1000.0
IF (exp(RelRhoOil) < 2.5) THEN
      D_t = max(0.01d0, (2.5d0 - exp(RelRhoOil)))
ELSE
      D_t = max(0.01d0, (exp(RelRhoOil) - 2.5d0))
END IF

!Viscosity transform (ranges checked)
IF (log(ViscOil) < 5.8) THEN
      V_t = 5.8 - log(ViscOil)
ELSE
      V_t = log(ViscOil) - 5.8                    !changed from 8.7 in text
END IF

!Saturate transform (ranges checked)
IF (Oil_Sat < 45.0) THEN
      S_t = 45.0 - Oil_Sat
ELSE
      S_t = max(eps, Oil_Sat - 45.0)
END IF

!Resin transform (ranges checked)
IF (ResinOil < 10.0) THEN
      R_t = max(0.1d0, 10.0 - ResinOil)
ELSE
      R_t = max(0.1d0, ResinOil - 10.0)
END IF

!Asphaltene transform (ranges checked)
IF (AsphOil < 4.0) THEN
      A_t = 4.0 - AsphOil
ELSE
      A_t = max(eps, AsphOil - 4.0)
END IF

!A/R ratio transform (ranges checked)
AR = AsphOil / ResinOil
IF (AR < 0.6) THEN
      AR_t = max(0.025d0, 0.6 - AR)
ELSE
      AR_t = max(0.025d0, AR - 0.6)
END IF

!calculate stabilityC (Eqn 15)
StabilityC = 12.3 + (0.259 * S_t) - (1.601 * R_t) - (17.2 * AR_t)                    &
                  - (0.5 * (V_t**3.0)) + (0.002 * (R_t**3.0))                        &
                  + (0.001 * (A_t**3.0)) + (8.51 * (AR_t**3.0))                      &
                  - (1.12 * log(V_t)) + (0.7 * log(R_t)) + (2.97 * log(AR_t))        &
                  + (6.0E-08 * (exp(V_t)**2.0)) - (1.96 * (exp(AR_t)**2.0))          &
                  - (4.0E-06 * (log10(D_t)/(D_t**2.0)))                              &
                  - (1.5E-04 * (log10(AR_t)/(AR_t**2.0)))

!determine emulsion state based on stabilityC (Table 10.3)
IF(StabilityC >= 2.2 .AND. StabilityC <= 15) THEN
      ClassIndex = 4                              !Stable
ELSEIF (StabilityC >= -12.0 .AND. StabilityC <= -0.7) THEN
      ClassIndex = 3                              !Mesostable
ELSEIF (StabilityC >= -18.3 .AND. StabilityC <= -9.1) THEN
      IF(RelRhoOil > 0.96 .AND. ViscOil > 6000.0) THEN
            ClassIndex = 2                        !Entrained
      END IF
ELSEIF (StabilityC >= -39.1 .AND. StabilityC <= -7.1) THEN
```

```
            IF (RelRhoOil < 0.85 .OR. RelRhoOil > 1.0) THEN
                IF (ViscOil < 100 .OR. ViscOil > 800000) THEN
                    IF(AsphOil < 1.0 .OR. ResinOil < 1.0) THEN
                        ClassIndex = 1                  !Unstable
                    END IF
                END IF
            END IF
        ELSE
            write(*,*)'Unresolved Emulsion Class - assume UNSTABLE'
            ClassIndex = 1
        END IF

        !Only update emulsion state if progressively more stable emulsions form
        IF (ClassIndex > StartClass) THEN

            StartClass = ClassIndex
            StartTime = ElapsedTime

            !determine formation time for emulsion state
            IF (ClassIndex >= 2)THEN
                FormTime = Class(ClassIndex, 7) + (Class(ClassIndex, 8) / ((SigWaveHeight*100.0)**1.5))      !(minutes)
            ELSE
                FormTime = 0.0
            END IF

            !calculate times to end of day, week and year
            !for Fingas values (from Table 10.4)
            FingasDay = StartTime + (FormTime * 60.0) + (1.0 * 86400.0)
            FingasWeek = StartTime + (FormTime * 60.0) + (7.0 * 86400.0)
            FingasYear = StartTime + (FormTime * 60.0) + (365.0 * 86400.0)

        END IF

    END IF

    !Calculate incremental increase in viscosity and water content per DT based on times above.
    IF (StartClass == 0) THEN
        WaterContent = WaterContent
        xViscEmul = 1.0
    ELSE
        IF (ElapsedTime <= FingasDay) THEN                          !emulsion time <= FormTime + 1 Day
            WaterContent= min(WaterContent + ((Class(StartClass, 1) / (FingasDay-StartTime)) * IDT),Class(StartClass,1))
            xViscEmul       = min(xViscEmul + (((Class(StartClass, 4) - 1.0) / (FingasDay-StartTime) ) * IDT),Class(StartClass,4))
        ELSEIF (ElapsedTime <= FingasWeek) THEN                     !Form Time + 1 Day <= emulsion time <= 1 Week
            WaterContent= max(WaterContent + (((Class(StartClass, 2) - Class(StartClass, 1)) /  &
                                (FingasWeek-FingasDay)) * IDT),Class(StartClass,2))
            xViscEmul       = min(xViscEmul + (((Class(StartClass, 5) - Class(StartClass, 4)) /  &
                                (FingasWeek-FingasDay)) * IDT),Class(StartClass,5))
        ELSEIF (ElapsedTime <= FingasYear) THEN                     !emulsion time <= 1 Year (+ FormTime)
            WaterContent= max(WaterContent + (((Class(StartClass, 3) - Class(StartClass, 2)) /  &
                                (FingasYear-FingasWeek)) * IDT),Class(StartClass,3))
            xViscEmul       = min(xViscEmul + (((Class(StartClass, 6) - Class(StartClass, 5)) /  &
                                (FingasYear-FingasWeek)) * IDT),Class(StartClass,6))
        ELSE
            WaterContent = Class(StartClass,3)
            xViscEmul = Class(StartClass,6)
        END IF
    END IF

    RETURN

    END SUBROUTINE Emulsify
```

Subroutine DISPERSE

```
!*****************************
!*     Subroutine Disperse   *
!*****************************
SUBROUTINE Disperse(n,ElapsedTime,FirstAP,AreaOil,ViscOil,RhoOil,pZ,MassOil,MassDisp)
!French-McKay, 2004
!"Oil Spill Impact Modelling: Development and Validation"
!Environmental Toxicology and Chemistry, Vol 23, No. 10, pp 2441-2456.
!after
!Delvigne, G., & Sweeney, C. 1988
!"Natural Dispersion of Oil"
!Oil and Chemical Pollution, Vol 4, pp 281-310

USE PARAM_MOD, ONLY:      SigWaveHeight,SigWaveLength,SigWavePeriod,idt,                    &
                    UWind_10,VWind_10,numpar,windwavemodel

IMPLICIT NONE

!I/O variables
INTEGER, INTENT(IN):: ElapsedTime,n
LOGICAL, INTENT(IN):: FirstAP(n)
double precision, intent(in):: AreaOil,ViscOil,RhoOil,MassOil
double precision, intent(inout):: MassDisp
DOUBLE PRECISION, DIMENSION(numpar), OPTIONAL, INTENT(INOUT) :: pZ

!Local variables
DOUBLE PRECISION:: Dbwe                                 !dissipated breaking wave energy per unit area (J/m2)
DOUBLE PRECISION:: Hbreak                               !breaking wave height (m)
DOUBLE PRECISION:: Cstar                                !empirical entrainment constant
DOUBLE PRECISION:: FracWave                                   !fraction of sea surface hit by breaking waves
DOUBLE PRECISION:: Oil_d50                              !mean oil droplet diameter (um)
DOUBLE PRECISION:: DropDiam(10)                                 !Do droplet diameter per size class (m)
DOUBLE PRECISION:: Qd(10)                               !Entrainment rate per size class (kg/m2s)
DOUBLE PRECISION:: Qdtotal                              !Total entrainment rate for all size classes
DOUBLE PRECISION:: DeltaDiam                                  !oil droplet interval diameter (m)
DOUBLE PRECISION:: Dmin                                 !minimum and maximum droplet diameter (m)
DOUBLE PRECISION:: Dmax                                 minimum and maximum droplet diameter (m)
DOUBLE PRECISION:: WindSpeed
INTEGER:: i                                     !droplet distribution array counter
DOUBLE PRECISION, PARAMETER:: Uth = 6.0                       !Threshold wind speed for the onset of breaking waves (m/s)
DOUBLE PRECISION, PARAMETER:: Ewave = 5000.0                  !mean energy dissipation rate per unit volume (J/m3-s) ranges between 1,000 and 10,000 (J/m3.s) - See Delvigne (1988)
DOUBLE PRECISION, PARAMETER:: FracOil = 1.0                   !fraction of sea surface covered by oil.
double precision :: pMass
integer:: pQd,pN
double precision:: ran,waveheight,waveperiod
logical :: found

IF (FirstAP(n)) THEN
      MassDisp = 0.0                                  !initialise mass dispersed
END IF

!Calculate empirical entrainment constant Cstar
IF (ViscOil < 132.0 ) THEN
      Cstar = exp( (-0.1023 * log(ViscOil)) + 7.572)
ELSE
      Cstar = exp( (-1.8927 * log(ViscOil)) + 16.313)
END IF
```

```
!calculate windspeed
WindSpeed = sqrt((Uwind_10**2.0) + (Vwind_10**2.0))

!determine waveheight and wave period
if(windwavemodel)then
      waveheight = SigWaveHeight                    !SWAN model output
      waveperiod = SigWavePeriod
else
      waveheight = 0.243 * (WindSpeed)**2.0 / Gravity                    !CERC formulation
      waveperiod = 8.13 * WindSpeed / gravity
end if

!calculate breaking wave height
Hbreak = (1.0/sqrt(2.0)) * waveheight

!Calculate dissipated breaking wave energy (J/m2)
Dbwe = 0.0034 * RhoWater * Gravity * (Hbreak**2.0)

!Calculate fraction of sea surface hit by breaking waves
IF (WindSpeed <= Uth ) THEN
      FracWave = 3E-06 * (WindSpeed**3.5 / WavePeriod)
ELSE
      FracWave = 0.032 * ( (WindSpeed - Uth) / WavePeriod)
END IF

!Calculate mean oil droplet diameter (um),
!minimum radius (m) and maximum radius (m)
Oil_d50 = 1818.0 * (Ewave**-0.5) * (ViscOil**0.34)                    !based on viscosity
Dmin = 0.1 * Oil_d50 * 1E-06                    convert from micrometers to meters)
Dmax = Oil_d50 * 1E-06                    !(convert from micrometers to meters)

!write(*,*)oil_d50,rmin,rmax
!Construct droplet size distribution
!Adopt 5No. size classes between Rmin and Rmax,
!equally spaced on diameter
DeltaDiam = (Dmax - Dmin ) / 5.0

!initialise total entrainment rate every timestep
Qdtotal = 0.0

DO i = 1, 5
      if(DropDiam(i) < 70e-6)then                    !70um
            !for each droplet interval, calculate centred droplet diameter, Do
            DropDiam(i) = ((2.0 * Rmin) + (0.5 * DeltaDiam)) + (DeltaDiam * (i-1))
            Qd(i) = Cstar * (Dbwe**0.57) * FracOil * FracWave * (DropDiam(i)**0.7) * DeltaDiam          !(kg/m2s)
      else
            Qd(i) = 0.0
      end if
      !sum over all droplet classes for total entrainment rate
      Qdtotal = Qdtotal + Qd(i)
END DO

!calculate mass dispered (kg) and volume dispersed (m3)
MassDisp = MassDisp + (Qdtotal * AreaOil * idt)

return

END SUBROUTINE Disperse
```

Subroutine DENSITY

```
!*******************************
!*     Subroutine Density     *
!*******************************
SUBROUTINE Density(RhoOil,MassEvap,MassSpill,WaterContent,DeltaRho)
!Buchanan, I. 1988
!"Methods for predicting the physical changes of oil spilled at sea"
!Oil and Chemical pollution vol 4(4) pp311-328
USE PARAM_MOD, ONLY:      WaterTemp,Evaporation,Emulsification,Oil_Dens, Oil_Dens_RefT

IMPLICIT NONE

double precision, intent(in)::MassEvap,MassSpill,WaterContent
double precision, intent(inout):: RhoOil
double precision, intent(out):: DeltaRho

!First re-calculate RhoOil based on temperature correction.
RhoOil = Oil_Dens *  (1.0 - CDensT * ((WaterTemp + 273.15) - Oil_Dens_RefT))                    !actual density of spilled oil at ocean temperature (kg/m3)

!Check if Evaporation enabled
!- if so include Evap effect on RhoOil already calculated
IF (Evaporation) THEN
      RhoOil = RhoOil * (1.0 + CDensE * (MassEvap / MassSpill))
END IF

!Check if Emulsification enabled
!- if so include Emuls effect on RhoOil already calculated
IF (Emulsification) THEN
      RhoOil = (WaterContent * RhoWater) + (RhoOil * (1.0-WaterContent))
END IF

DeltaRho  = (RhoWater - RhoOil) / RhoWater

RETURN

END SUBROUTINE Density
```

Subroutine VISCOSITY

```
!*******************************
!*     Subroutine Viscosity     *
!*******************************
SUBROUTINE Viscosity(ViscOil, RhoOil, MassEvap, MassSpill, xviscemul)

USE PARAM_MOD, ONLY: Dyn_Visc,Kin_Visc,WaterTemp,Dyn_Visc_RefT, Kin_Visc_RefT,Evaporation,Emulsification

IMPLICIT NONE

double precision, intent(in):: RhoOil, MassEvap,MassSpill,xviscemul
double precision, intent(inout):: ViscOil

!NewViscosity   = RefViscosity * exp(dViscTemp + dViscEvap) * xViscEmul
!              = RefViscosity * exp(dViscTemp) * exp(dViscEvap) * xViscEmul
 IF (Dyn_Visc > 0.0) THEN
      ViscOil = 1000.0 * Dyn_Visc * exp(ViscCt * ((1.0/(WaterTemp+273.15)) - (1.0/Dyn_Visc_RefT)))          !Dynamic (kg/ms -> cP)
 ELSE
      ViscOil = 1000.0 * Kin_Visc * exp(ViscCt * ((1.0/(WaterTemp+273.15)) - (1.0/Kin_Visc_RefT)))          !Kinematic (m2/s -> cSt)
      ViscOil = ViscOil * RhoOil                                    !Dynamic (cP)
END IF
```

```
IF (Evaporation) THEN                                   !(Mackay,1980)
      ViscOil = ViscOil * exp((Evap_C4 * (MassEvap / MassSpill)))
END IF

IF (Emulsification) THEN                                !(Fingas 2011)
      ViscOil = ViscOil * xViscEmul
END IF

RETURN

END SUBROUTINE Viscosity
```

Subroutine DISSOLUTION

```
!*********************************
!*    Subroutine Dissolution   *
!*********************************
SUBROUTINE Dissolution(ElapsedTime,WaterContent,AreaOil,MassDiss)
!Cohen, Y., D. Mackay and W.Y. Shiu, (1980):
!"Mass Transfer Rates Between Oil Slicks and Water".
!The Canadian Journal of Chemical Engineering. Vol. 58.
USE PARAM_MOD, ONLY: idt

IMPLICIT NONE

INTEGER, INTENT(IN):: ElapsedTime
double precision, intent(in):: WaterContent,AreaOil
Double precision, INTENT(INOUT):: MassDiss

DOUBLE PRECISION, parameter:: InitSol = 0.03            !need to change this if data become available
DOUBLE PRECISION, parameter:: DecayRate = 0.1               !need to change this if data become available
DOUBLE PRECISION, parameter:: MassTranCoeff = 0.01              !(m/hr)      !need to change this if data become available
DOUBLE PRECISION:: Sol

Sol  = InitSol * exp(-DecayRate * (ElapsedTime/3600.0))         !convert ElaspedTime from seconds to hours
MassDiss = MassDiss + (MassTranCoeff/3600.0) * (1.0-WaterContent) *      &
                AreaOil * Sol * idt
RETURN

END SUBROUTINE Dissolution
```