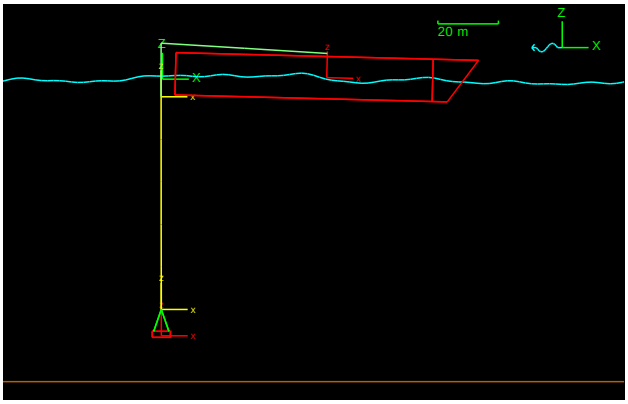
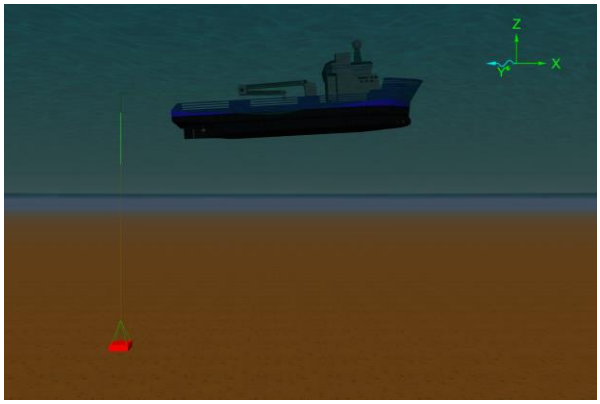
F03 PID controlled active winch

This package-lowering example uses a winch to model an active heave compensation (AHC) device. An *external function*, written in Python, is used to control the length of the winch, thereby holding an object at a particular depth.





An OrcaFlex line object is used to model the majority of the work wire. Using a line means that the wire's physical and hydrodynamic properties are accounted for. The winch between the vessel and work wire is controlled with a PID controller via the OrcaFlex application programming interface (OrcFxAPI).

Although Python has been used to write the controller code in this example, C++ or Delphi could have been used instead.

To view this example, you will need a suitable Python installation. Note, the OrcaFlex installation package (11.3 onwards) includes the option to install a compact embedded Python distribution. If this option is selected when installing OrcaFlex, a separate installation of Python is not required for embedded Python features e.g. Python external functions and user defined results. The Python interface | Installation page of the OrcFxAPI help documentation provides some further guidance about this.

Building the model

The package being lowered is a simple 6D *lumped buoy*, with links used to represent the lift rigging. An OrcaFlex line is used between the links and the lowering winch. Links and winches have no mass, nor can they receive hydrodynamic forces. So, using a winch for the entire work wire would mean that any hydrodynamic effects, acting on the wire, would be missed.

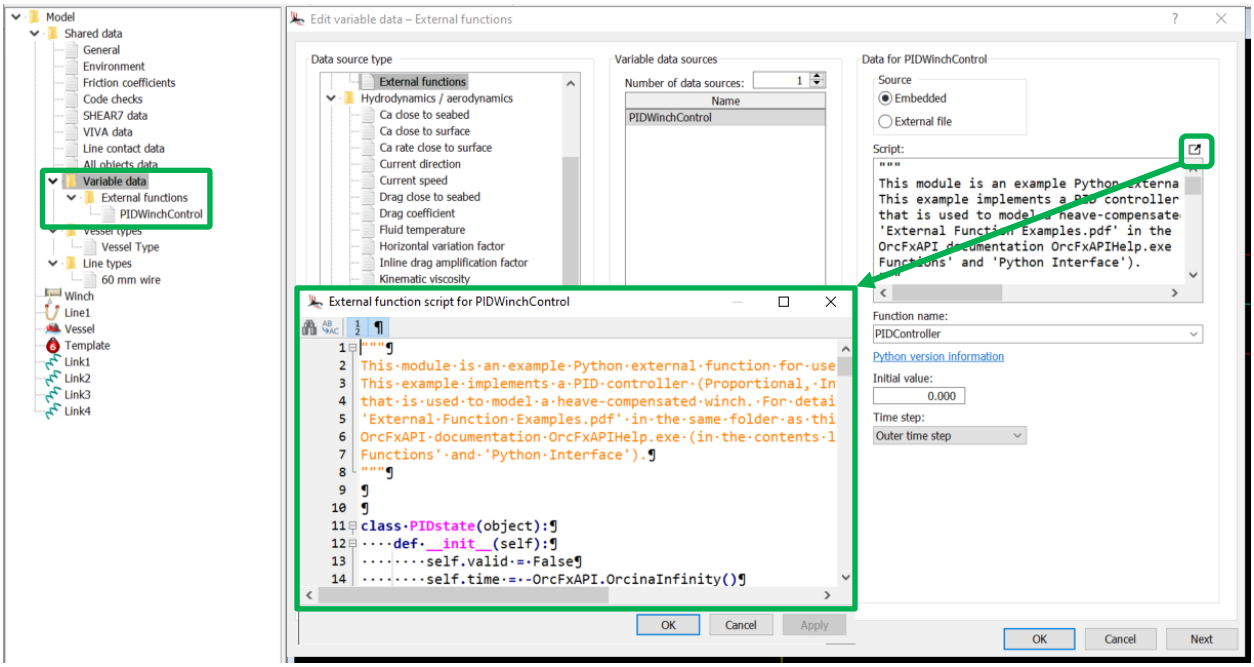
The vessel uses displacement RAOs, and so moves in response to the wave loading imposed on the system. Our aim is to use the winch to compensate for the vessel motion and maintain the package at a constant depth, i.e. active heave compensation.

To control the winch, a PID controller (proportional–integral–derivative controller) has been implemented. This is a generic control-loop feedback mechanism, which is widely used in control systems. The PID control algorithm has been coded in Python, a dynamic scripting language, and interfaces with the model via the OrcFxAPI.

Further details about the Python interface to OrcaFlex can be found in the dedicated OrcFxAPI help documentation.

The Python code can either exist in a separate Python file (.py), or it can be embedded in the OrcaFlex model. In this case, we have embedded the script in the model. To view the code, double-click on the *PIDWinchControl* dataset in the *model browser*, beneath the *Variable data → External functions* folder. The script is visible in the *script* box, however clicking on the *pop out editor* button opens the script in a more user-friendly text editor.

Closing the text editor and returning to the *variable data* form, the name of the function within the Python script – used to control the winch – has been selected from the *function name* drop-down box (OrcaFlex queries the contents of the Python file automatically and lists the available functions).



We also need to define an *initial value*, i.e. the value passed to the external function the first time it is called, and how regularly the external function should be called. In this case, we are using the implicit integration scheme, so the function will be called on each iteration of each time step regardless of what has been selected in the *time step* drop-down box.

Once the external function is set up, we can use it to control the winch. This is done on the *Winch* data form, *control* page. Here, the name of the external function, *PIDWinchControl*, is used as a variable data item for the winch *payout rate* throughout the dynamic simulation.

On the *tags* page, additional information that needs to be passed to the external function has been listed. In this example, the *tags* are used to tell the function the following information:

- *ControlledObject* – the name of the object to be controlled (*Template*).
- *ControlledVariable* – the name of the object result to be controlled (*Z*).
- *ControlStartTime* – the control start time, which is set to 20s so that the motion of the package, with and without heave compensation, can be compared.
- *TargetValue* – the target depth at which the package should be held (85m), so Z = -85.
- *MinValue* / *MaxValue* – the minimum and maximum payout rate for the winch (optional).
- *k0*, *kP*, *kI* and *kD* – the PID control parameters.

Results

Opening the simulation file automatically opens the default workspace, which displays some important results of interest.

The variation in winch *tension* is shown (top right) to illustrate the loads experienced during the motion. The *Template* submerged mass is 19.75 te, and so a mean tension of approx. 200 kN is reasonable. There is a shock load on the winch when it begins to control the package depth at time 20s. This is because of a step change in acceleration.

Time histories of winch *length* and template *Z* position are shown below the tension plot; the latter clearly showing the effect of the PID controller holding the structure close to the required depth. Run the replay (*Ctrl + R*) to see this in action in the wire frame model view.

Vessel motion near the stern is also plotted (top left), and has an approximate range of up to ±5 m. The corresponding range in template depth, once the heave compensation is operating, is less than ±0.5 m. The effectiveness of the compensation depends on the PID controller parameters, but the maximum capability of the real equipment should be considered when building a model like this i.e. the parameters must be 'tuned' to reflect the performance of the real system. If this is not done, then it is possible to achieve better or even worse performance in a simulation than may be realistic.

Note: further external function examples, in both Python and C++, are available for download from the external function examples page of the Orcina website. An introduction to the Python interface to OrcaFlex can be found on the OrcaFlex documentation page.