Chapter 2
www.FreeDyn.at – A free, GUI based multi-body dynamics software for stand-alone use and with C API for linking it to Python, Matlab or custom code

Wolfgang Witteveen and Stefan Oberpeilsteiner

Abstract FreeDyn is a free multibody simulation (MBS) package developed by the Upper Austrian University of Applied Sciences. The precompiled Windows binaries can be downloaded from www.freedyn.at. It features a graphical user interface (GUI) and a C++ solver which is a modified version of the HHT time integration method. FreeDyn can be used as standalone software via its GUI or can be linked to other software such as Python, MATLAB, Scilab or custom codes through a C-Interface.

The new Python interface makes FreeDyn particularly attractive to universities and research groups. Researchers can outsource multi-body simulations to FreeDyn while addressing unique research questions using Python to call FreeDyn functions. FreeDyn supports rigid and flexible bodies, force elements and constraints. Users can define state and time-dependent force and constraint elements using mathematical expressions. Additionally, advanced force elements may be linked to FreeDyn using a user written subroutine (DLL).

The extended abstract contains a brief review of the underlaying theoretical concepts, a short summary of all modelling elements and of all callable FreeDyn functions. Furthermore, a simple example will be given in detail to demonstrate the binding of FreeDyn to Python.

Keywords Multibody Dynamics · Free Software · Python

Introduction and Theoretical Background

FreeDyn [1] is a freely available Windows® software package for the pre- and post-processing and numerical time integration of multibody dynamic systems. A user interface (UI) supports the intuitive modelling of complex models via typical graphic oriented methods. The final model can be exported into a solver for the numerical time integration. The results can be read in again, animated and plotted. The automatically generated equations of motion take on the form

$$\begin{aligned} \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C_q}^{\mathbf{T}}(\mathbf{q})\lambda &= \mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}) \\ \mathbf{C}(\mathbf{q}, \mathbf{t}) &= \mathbf{0} \end{aligned} \tag{1}$$

where the state vector $\mathbf{q}$ contains the degrees of freedom (dof) of the system. It consists of n sub-vectors containing the dof of the n modeled bodies. The translation of a rigid body is characterized by the 3 coordinates of the center of mass. For the parameterization of the rotation, Euler parameters are chosen. The matrix $\mathbf{M}$ is the state dependent mass matrix with n block diagonal entries according to the mass matrices of the individual bodies. The vector $\mathbf{Q}$ contains the generalized forces and the quadratic velocity vector. The constraint equations are collected in the vector $\mathbf{C}$ and depend on the state vector $\mathbf{q}$ and the time t. The constraint forces are computed via the constraint Jacobian ${\mathbf{C}_q}^T$ and the vector of the Lagrange multipliers $\boldsymbol{\lambda}$. Readers who are more interested can find more details in the following list:

- On the numerical time integration via an HHT solver see [2].

Wolfgang Witteveen
University of Applied Sciences Upper Austria, Degree Program Mechanical Engineering, 4600 Wels, Austria
e-mail: wolfgang.witteveen@fh-wels.at

Stefan Oberpeilsteiner
principia MBS GmbH, 4113 St. Martin, Austria, www.principia-mbs.com
e-mail: stefan@principia-mbs.com

Ludovic Renson et al. (eds.), *Nonlinear Structures & Systems, Vol. 1*, Conference Proceedings
of the Society for Experimental Mechanics Series, https://doi.org/10.13052/97887-438-0146-7_2

- On known problems concerning numerical damping in connection with Euler parameters see [3].
- On flexible bodies see [4] and [5].
- Additional publications on different theoretical and application-oriented topics see [6], [7] and [8].

Available Modelling Elements

Bodies:

- Rigid bodies can either be selected from a library of simple geometries (Sphere, Cone, Cylinder, Box) or imported via stl or obj file.
- The representation of flexible bodies is based on the Floating Frame of Reference Formulation (FFRF). A text file is required for the import. If an FE solver does not support the implemented format, a corresponding parser can be written.

Data Objects:
Data objects can be used to specify parameters and characteristic curves that can be referenced by other FreeDyn elements. This allows the parametrization of a model.

- A parameter consists of a name and a value. It can be referenced from other FreeDyn elements via its name which will be replaced by the value when numerical time integration starts.
- A spline can be used to describe a relationship between two quantities. It is defined by certain numbers of interpolation points. Once defined, the spline can be used with special access functions in force elements and motions.

Measures:

- Translational measures can be used to measure the displacement or velocity between two points of two bodies. In case of a measure between a body and ground the acceleration can be measured as well.
- Rotational measures can be used to measure the rotational displacement or rotational velocity between two vectors of two bodies.

Constraints:
Constraints restrict the motion possibilities of two bodies relative to each other. One of the two bodies may be ground.

- A generic constraint restricts one motion along or around one axis. By combining several such generic constraints, arbitrary or well-known constraints such as “fixed” or “spherical” can be realized.
- Motion constraints can be used to impose a certain motion on a body. The type and characteristics of the motion can be defined with measures, common mathematical expressions and the former mentioned splines.
- A gear constraint can be used to link the rotations of two bodies via a gear ratio.

Forces:

- Generic forces/torques can act between two bodies or between one body and ground. Measures, data objects and mathematical expressions are available for a wide range of modelling possibilities of state and time dependent forces and torques.
- With the external subroutine force element, arbitrary self-written forces can be linked to FreeDyn via a dll. With this interface, it would also be conceivable, for example, to integrate force elements that are written in Python.
- The tire force element provides a Pacejka tire model which acts between a rotating body and a street profile.
- Force contact offers the possibility to define a contact model between two rigid bodies based on the work and code published by G. Hippmann, see [7].

Stand-Alone Software or C-Api

Figure 1 illustrates how FreeDyn can be used. The left-hand side indicates the data flow for the use as a stand-alone tool. The user mainly interacts with the graphical UI. Once the model has been built, the numerical time integration can be started via a corresponding dialog box and the results are automatically fed back into the UI.

The right-hand side of Figure 1 illustrates the use of the C-API. The graphical UI can still be used to build up a model and investigate the results. Once the model has been exported (= model file) it can be referenced via functions calls of the C-API. The FreeDyn model can then be used and manipulated inside self-written program code like Python, Matlab, Scilab or custom code. Furthermore, it can be used as a block inside a Matlab/Simulink or Scilab/XCos simulation. For each current time integration point, all kind of information can be requested from the solver. These are for example the displacements, velocities, accelerations, measures, constraint forces, Lagrange multipliers, constraint Jacobian, Jacobian of the full system and other quantities. All this data can be used of optimization tasks or whatever is needed. The interface can also be used to modify model parameters. These are e.g. integration parameters, model states or forces. To give an impression of the possibilities offered by the C-API, some of the available commands are listed and explained in more detail:

- The function call `createFreeDynModel(...)` creates a FreeDyn model based on a model file name.
- `setModelAsActiv(...)` gives the possibility to select a specific FreeDyn model. This is of particular importance when multiple models have been created.
- `getModelInfos(...)` returns information like the number of degrees of freedom, the number of Lagrange multipliers and others.
- With `getMeasureValueWithMeasureName(...)` the content of a measure can be requested.
- The function call `solveTimeInterval(...)` allows to solve for a time interval. This interval can be very long, but also be very short, such as one single integration time step.
- `getPhysicalDofRelatedVector(...)` provides all kind of internal vectors like the quadratic velocity forces, the acceleration inertia forces, the sum of external applied forces, and others more.
- `getModelRelatedMatrix(...)` provides all kind of internal matrixes like the state depended mass matrix, the derivation of the external forces with respect to the generalized coordinates, the constraint Jacobian and others more.

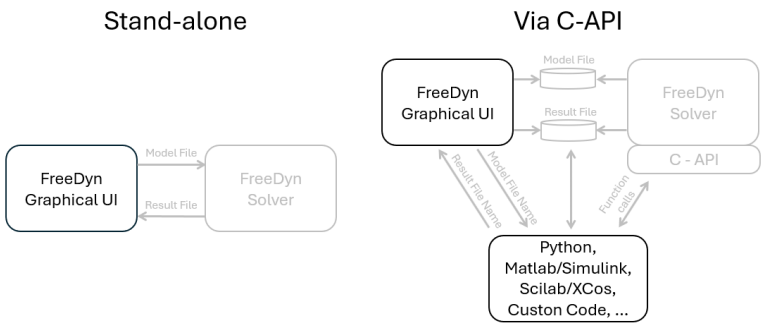

Fig. 1 FreeDyn use as stand-alone tool or via C-API

Python Example

Part of the latest release is the file *fdApi.py*. It can be included in Python and contains an interface to FreeDyn which can be controlled and used by the commands defined in this file. The functionality of this interface is explained using a simple demonstration example. The excitation force of a single-mass oscillator f(t) is searched via a so called "virtual iteration" in such a way that the mass follows a given motion x(t). The virtual iteration is a proven method of inverse dynamics and can be found in detail in section 3 of [13]. The method works in the frequency domain and requires the frequency response (FRF) functions between all outputs and inputs. In an iterative process, the input is changed in the frequency domain until the deviation of the output from the target curve is small enough. The input and output variables are transformed back and forth between the time and frequency domain, as the update based on the error takes place in the frequency domain, but the simulation takes place in the time domain. The single-mass oscillator was modeled in FreeDyn and the model file was stored accordingly. We assume that the FRF between the input f(t) and the output x(t) is already available. In the following the Python code is given for this example. The commands related to the FreeDyn API are highlighted with yellow color. Fig. 2 holds the result. It can be seen that the mass follows the specified motion.

```
import numpy as np
import math
import sys
import os
import matplotlib.pyplot as plt
from ctypes import *
import utils
#-------------------------------------------------------------------------------------
# Initialize FreeDyn API
# Add FreeDyn API to the system path
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..', '..', 'fdApi')))
import fdApi
fdApi.init()
#-------------------------------------------------------------------------------------
# Initialize FreeDyn model
# Define filename+path of FreeDynModel
fds_file_path = os.path.abspath(os.path.join(os.path.dirname(__file__), "singleMassOscillator.fds"))
# Create and set model as active
model_index = fdApi.createModel(fds_file_path, 'no')
if model_index < 0:
    sys.exit("Error: Failed to create model")
fdApi.setModelAsActive(model_index)
states = fdApi.generateStateVectors()
#define name of variable to be modified (excitation force)
spline_label = "forceSpline"
#-------------------------------------------------------------------------------------
# Prepare time vector according to model
v_time = utils.getTimeVector()
n_time_steps = v_time.size
#-------------------------------------------------------------------------------------
# Load transferfunction and target Y-coordinate of single mass oscillator
H = utils.getTransferFunction(v_time)
Y_target = utils.getTargetSignal(v_time)
y_target = np.fft.irfft(Y_target, n_time_steps)
#-------------------------------------------------------------------------------------
# Initialize variables
# compute first shot using transferfunction and target of cg y
U = np.zeros_like(Y_target, dtype=np.complex128)  # Change dtype to complex128
for i, Yi in enumerate(Y_target):
    U[i] = Y_target[i] / H[i]
# Iteration parameters
it = 1; i_max = 1000; alpha = 1.0; eps = 0.001; e_old = 0.0; alpha_min = 1e-10
#-------------------------------------------------------------------------------------
# Iteration loop
while True:
    # Update force on FreeDyn model
    u = np.fft.irfft(U,n_time_steps)
    spline_data = np.vstack((v_time, u)).T
    fdApi.modifySpline(spline_label, spline_data)
    # Simulate the model again
    if fdApi.solveEoM() < 0:
        sys.exit("Error: Simulation failed after modifying spline")
    # Copy generalized coordinates
    y = np.zeros(n_time_steps)
    for i in range(n_time_steps):
        fdApi.getStatesAtTimeIndex(i, states["Q"])
        y[i] = states["Q"][1, 0]
    Y = np.fft.rfft(y)
    # Compute error
    error = np.linalg.norm(y - y_target)
    utils.updatePlots(y,u, it, error)
    if error < eps:
        print("Converged: Error threshold reached")
        break
    elif it >= i_max:
        print("Failed to converge: Maximum iterations reached")
        break
    elif alpha < alpha_min:
        print("Failed to converge: Alpha below minimum")
        break
    # Update U and H
    if it > 1 and e_old < error:
```

```
        # Reduce alpha if the new result is not better
        U = np.copy(U_old)
        alpha /= 2.0
    else:
        U_old = np.copy(U)

    for i, Ui in enumerate(U):
        U[i] += alpha * (Y_target[i] - Y[i]) / H[i]
    # Store previous error value
    e_old = error
    #increment iteration counter
    it += 1
# Clean up and close
fdApi.deleteModel(model_index)
plt.ioff()
plt.show()
```

Additonal Examples

To give interested readers an even better idea of FreeDyn, some additional projects are listed below with corresponding references.

- In [8] an iterative coupling procedure of experimental and numerical subsystems to one overall system is introduced. The numerical subsystem is an offroad vehicle, which is modelled with FreeDyn, see the upper left picture in Fig. 3.
- In [10] FreeDyn is used for the development of motorcycle rider assistance systems. The motorcycle is modelled with FreeDyn and used as block in a Matlab/Simulink model. The upper right picture of Fig. 3 holds a screenshot of the Matlab/Simulink model.
- [11] reports on an optimization problem in which energy-optimal control of a robot motion are sought. A picture of the FreeDyn robot model is given in the lower right corner of Fig. 3.
- In [12], rolling processes of tooth flanks (lower left image of Fig. 3) are investigated under consideration of measured surface roughness.

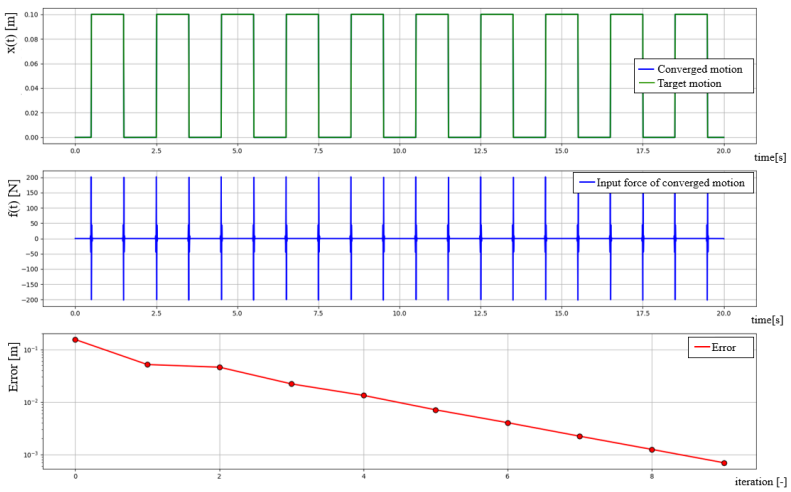

Fig. 2 Result of the virtual iteration.

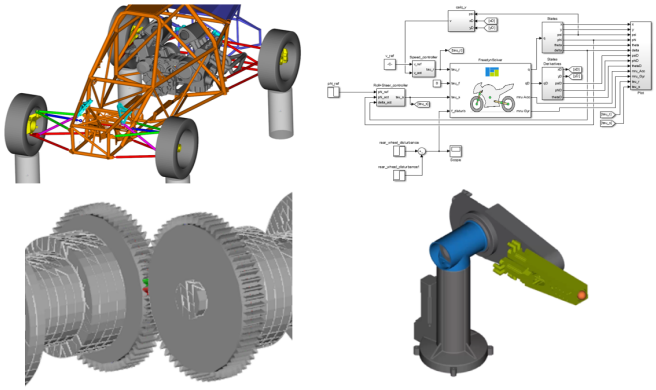

Fig. 3 FreeDyn examples.

References

1. www.freedyn.at
2. Negrut, D., Rampalli, R., Ottarsson, G., Sajdak, A., 2007, "On an Implementation of the Hilbert-Hughes-Taylor Method in the Context of Index 3 Differential-Algebraic Equations of Multibody Dynamics", Journal of Computational and Nonlinear Dynamics, 2, pp. 73–85, DOI:10.111./1.2389231
3. Sherif, K., Nachbagauer, K., Steiner, W., Lauß, T.: A modified HHT method for the numerical simulation of rigid body rotations with Euler parameters - Multibody System Dynamics, Vol. 46, No. 2, 2019
4. Sherif, K., Nachbagauer K., 2014, "A detailed derivation of the velocity-dependent inertia forces in the floating frame of reference formulation", Journal for Computational and Nonlinear Dynamics, doi:10.1115/1.4026083 (8 pages), 2014.
5. Witteveen, W., Pichler, F., 2019, "On the relevance of inertia related terms in the equations of motion of a flexible body in the floating frame of reference formulation", Multibody System Dynamics. 46, 1, S. 77–105
6. Sherif, K., Nachbagauer, K., Steiner, W., 2015, "On the rotational equations of motion in rigid body dynamics when using Euler parameters", Nonlinear Dyn 81, 343–352, https://doi.org/10.1007/s11071-015-1995-3
7. Witteveen, W., Thomas, L., Stefan, O., 2021, "FreeDyn: A Free, Flexible and State of the Art Multibody Simulation Package for Education, Research and Industrial Applications", In: Kerschen, G., Brake, M.R., Renson, L. (eds) Nonlinear Structures & Systems, Volume 1. Conference Proceedings of the Society for Experimental Mechanics Series. Springer, Cham. https://doi.org/10.1007/978-3-030-47626-7_8
8. Witteveen, W., Koller, L, 2023, "FRF-based non-simultaneous real-time hybrid testing of multiple subsystems with moderate nonlinearities", Mechanical Systems and Signal Processing, Volume 188, 109944, https://doi.org/10.1016/j.ymssp.2022.109944.
9. Hippmann G., Modellierung von Kontakten komplex geformter Körper in der Mehrkörperdynamik, PhD Thesis, Institut fur Mechanik und Mechatronik, TU Wien, 2004 (http://pcm.hippmann.org/doc/hippmann_dissertation.pdf,cited09.2024)
10. Haas, S., Dück, M., Winkler, A., Grabmair, G. et al., (2020) "Free Multibody Cosimulation Based Prototyping of Motorcycle Rider Assistance Systems", SAE Technical Paper 2020-32-2306,https://doi.org/10.4271/2020-32-2306.
11. T. Lauß, S. Oberpeilsteiner, K. Sherif and W. Steiner, "Inverse Dynamics of an Industrial Robot Using Motion Constraints," *2019 20th International Conference on Research and Education in Mechatronics (REM)*, Wels, Austria, 2019, pp. 1–7. doi:10.1109/REM.2019.8744124
12. Grünberger, J., Steiner, W., Witteveen, W. (2024), „Multibody Simulation of Gearbox Dynamics using 3D Contact Modeling and measured Tooth Geometries.", In Proceedings of 3rd International Symposium on Industrial Engineering and Automation Article 112
13. S. Reichl, 2011, Inverse Dynamics and Trajectory Tracing, Sündwestdeutscher Verlag für Hochschulschriften, ISBN 9783838121697