Marine

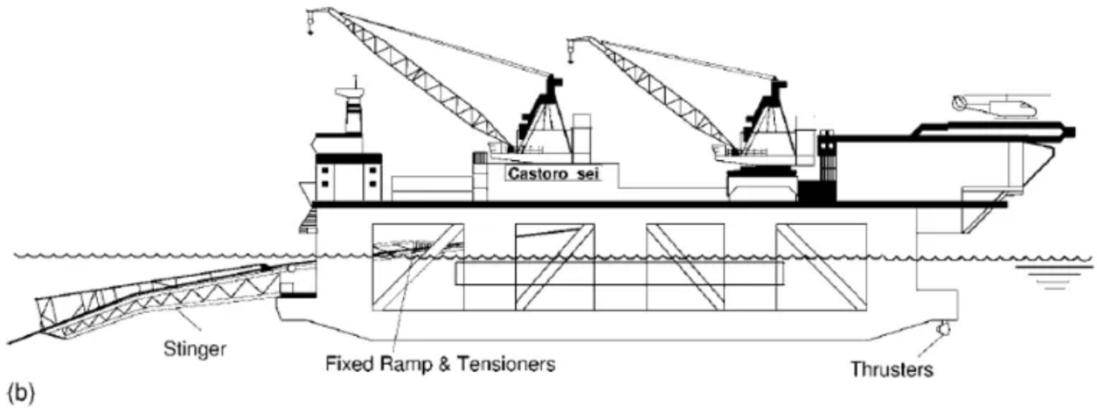
This is a manual for marine & offshore engineering software developed in the years of 2021 - 2025.

Table of contents:

1. PDE pipe

1.1. Background and rationale

Subsea pipelines are essential for transporting oil and gas from offshore fields to onshore facilities or between platforms. Installing subsea pipelines is a complex and challenging process that requires careful planning, design, and execution. There are different methods for installing subsea pipelines, depending on the water depth, pipe diameter, seabed conditions and environmental factors. One of the most common methods is S-lay, where the pipe is welded and coated on board a vessel and then lowered to the seabed in an S-shaped curve.



In this paper, I took approach as per thesis in [1]. I have used what the author called — a PDE pipe model in 6 degrees of freedom (DoF) per node, i.e. surge, sway, heave, roll, pitch, and yaw.

1.2. Methodology

1.2.1. Finite element model

Finite element method **(**FEM**)** is a widely used method for numerically solving differential equations arising in engineering and mathematical modelling. The FEM is a general numerical method for solving partial differential equations in two or three space

variables. To solve a problem, the FEM subdivides a large system into smaller, simpler parts that are called finite elements. This is achieved by a particular space discretization in the space dimensions, which is implemented by the construction of a mesh of the object: the numerical domain for the solution, which has a finite number of points. The finite element method formulation of a boundary value problem finally results in a system of algebraic equations. The method approximates the unknown function over the domain. The simple equations that model these finite elements are then assembled into a larger system of equations that models the entire problem. The FEM then approximates a solution by minimizing an associated error function via the calculus of variations (source Wikipedia).

For this model, space frame element was used,[2].

1.2.2. Dynamics

The linear and angular momentum balance equations for a nonlinear elastic beam are derived from Simo (1985).

1.2.3. Numerical

Odeint (solve_ivp) solver from scipy was used.

1.3. Model and implementation

The pipe at one end was fixed to sea floor and the other end was attached to center of gravity (CoG) of the vessel.

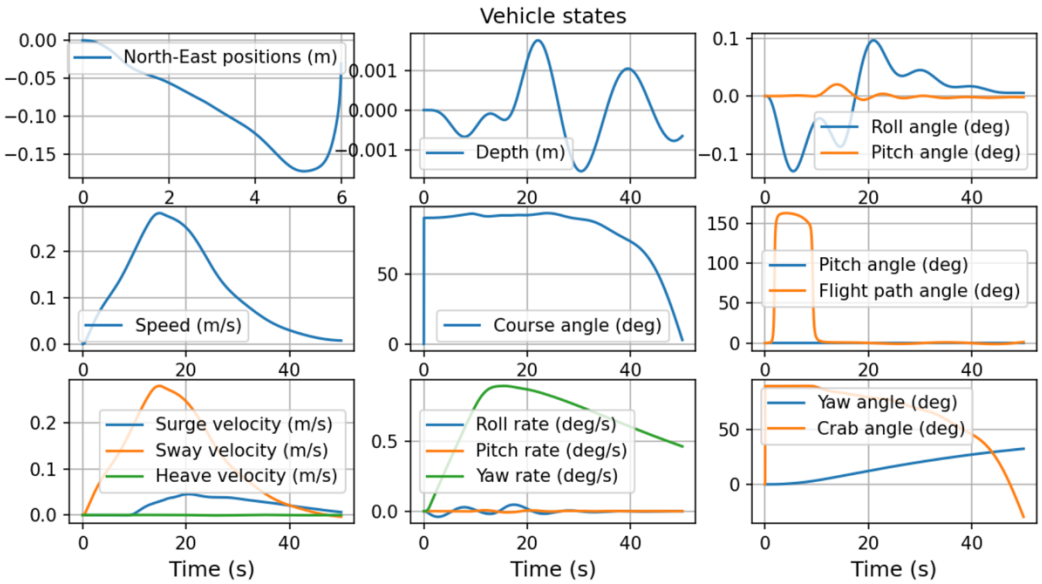
Some steps of the modelling are described below:

a) Catenary equation for an initial pipe profile
b) Static solution on top of catenary

In [100…

```
def static_func(Q): # It is not working properly for non zero roll, pitch and yaw; serves our needs though
    t=0

    x,y,z=Q[0:node_N],Q[2*node_N:3*node_N],Q[4*node_N:5*node_N]
    dx,dy,dz=Q[1*node_N:2*node_N],Q[3*node_N:4*node_N],Q[5*node_N:6*node_N]
    ϕ,θ,ψ=Q[6*node_N:7*node_N],Q[8*node_N:9*node_N],Q[10*node_N:11*node_N]
    Ret = Re_t(ϕ,θ,ψ)


    dϕ,dθ,dψ=Q[7*node_N:8*node_N],Q[9*node_N:10*node_N],Q[11*node_N:12*node_N]

    η=[x[-1],y[-1],z[-1],ϕ[-1],θ[-1],ψ[-1]]

    tau_force=np.array([-Fx_0,0,0,0,0,0])

    Z0= - np.dot(Re_b(Ret),tau_force[:3])
    Z1 = d_s(ne, x, y, z, Ret,  ωe(ϕ,θ,ψ,None,None,None)).T

    DT_0=np.zeros((3,3))
    Z2=ne_(dx,dy,dz,DT_0,Ret).T


    Z =  Z1 + Z2 + Z0

#####################################################

    ddx,ddy, ddz = np.linalg.solve(Ws[0]*Re_b(Ret),Z).T
    ddx,ddy, ddz = np.einsum('ijk,ik->ij',Ret, np.stack([ddx, ddy,ddz], axis=0).T).T

#####################################################



    C0=np.einsum('ijk,kp->ijp',np.einsum('ijk,ikr->ijr',np.linalg.inv(Π(ϕ,θ,ψ).astype(np.float64)),
                        Re_b(Ret)),tau_force[3:].reshape(3,1))

    C3=d_me(ϕ,θ,ψ,Ret,  ωe(ϕ,θ,ψ,None,None,None))

    B_ =C3
    B=(B_ + C0.squeeze())

    A2 = Irho_e(Ret,Irho).astype(np.float64)
    A3=Π(ϕ,θ,ψ).astype(np.float64)
    A=np.einsum('ijk,ikr->ijr',A2,A3)


#########################################
    ans_ϕ=[]
    ans_θ=[]
    ans_ψ=[]
    for i in range(len(A)):
        a,b,c=np.linalg.lstsq(A[i],B[i], rcond=None)[0]
        ans_ϕ.append(a)
        ans_θ.append(b)
        ans_ψ.append(c)

    ddϕ,ddθ,ddψ=np.array(ans_ϕ),np.array(ans_θ),np.array(ans_ψ)
#########################################

    ans=np.concatenate([dx, ddx, dy, ddy, dz, ddz, dϕ, ddϕ, dθ, ddθ, dψ, ddψ], axis=0)


    return ans
```

c) Vessel motion was produced by Python Vessel Simulator,[3]:



d) Created FEM model to calculate the force along the pipe.

e) Dynamic model:

In [137]:

```
def grayscott1d(t, Q, coefs, T):

    x,y,z=Q[0:node_N],Q[2*node_N:3*node_N],Q[4*node_N:5*node_N]

    idx = find_nearest(np.array(ans_t),  t)

    x_ve = simData[:, 0][idx]
    y_ve = simData[:, 1][idx]
    z_ve = simData[:, 2][idx]
    phi_ve = ssa(simData[:, 3][idx])
    theta_ve = ssa(simData[:, 4][idx])
    psi_v = ssa(simData[:, 5][idx])



    u_ve = simData[:, 6][idx]
    v_ve = simData[:, 7][idx]
    w_ve = simData[:, 8][idx]
    p_ve = simData[:, 9][idx]
    q_ve = simData[:, 10][idx]
    r_ve = simData[:, 11][idx]


    dx,dy,dz=Q[1*node_N:2*node_N],Q[3*node_N:4*node_N],Q[5*node_N:6*node_N]

    dφ,dθ,dψ=Q[7*node_N:8*node_N],Q[9*node_N:10*node_N],Q[11*node_N:12*node_N]


    vessel_movement = np.zeros(6).astype(np.float64)

    del_t = abs(t-T.my_t)

    vessel_movement[0] = u_ve*del_t
    vessel_movement[1] = v_ve*del_t
    vessel_movement[2] = -w_ve*del_t
    vessel_movement[3] = p_ve*del_t
    vessel_movement[4] = q_ve*del_t
    vessel_movement[5] = r_ve*del_t

    fors, mom, angle = PipeForce(x, y, z, vessel_movement, coefs)

#     φ,θ,ψ=angle.T
    Ret = Re_t(φ,θ,ψ)

    Z0= - np.einsum('ijk,ik->ij', Re_b(Ret), fors).squeeze()

    Z1=d_s(ne, x, y, z, Ret,  ωe(φ,θ,ψ,None,None,None)).T

    Z2=ne_(dx,dy,dz,DT,Ret).T


    Z= (
        Z1
        + Z2
        +Z0
       )

############################################################
    RRRR=Re_b(Ret)


    ddx,ddy, ddz = np.linalg.solve(np.einsum('i,ijk->ijk',M_t,Re_b(Ret)),Z).T

    ddx,ddy, ddz = np.einsum('ijk,ik->ij',Ret, np.stack([ddx, ddy,ddz], axis=0).T).T

############################################################


    C1=np.einsum('ijk,ik->ij', Irho_e(Ret,Irho).astype(np.float64),
                                                    np.einsum('ijk,ik->ij', Π(dφ,dθ,dψ).astype(np.float64),
                                                              np.array([dφ,dθ,dψ]).astype(np.float64).T))

    test=np.einsum('ijk,ik->ij',Π(φ,θ,ψ).astype(np.float64),np.array([dφ,dθ,dψ]).astype(np.float64).T)

    C2= np.cross(
         test,
                 np.einsum('ijk,ik->ij',Irho_e(Ret,Irho).astype(np.float64),
                           test))


    C3=d_me(φ,θ,ψ,Ret, ωe(φ,θ,ψ,None,None,None))

    C4= np.cross(d_s(phi,x,y,z,Ret, ωe(φ,θ,ψ,None,None,None)).T, ne(x,y,z,Ret, ωe(φ,θ,ψ,None,None,None),None))

    K1=test.T

    C5= np.einsum('ijk,ik->ij',Ret, -np.dot(DR,K1).astype(np.float64).T)

    C0=-np.einsum('ijk,ik->ij', np.einsum('ijk,ikr->ijr',np.linalg.inv(Π(φ,θ,ψ).astype(np.float64)),
                          Re_b(Ret)), mom ).squeeze()
    B_ =(
        -C1
        -C2
        +C3
        +C4
        +C5
    )
    B= B_ +C0

    A2 = Irho_e(Ret,Irho).astype(np.float64)
    A3=Π(φ,θ,ψ).astype(np.float64)
    A=np.einsum('ijk,ikr->ijr',A2,A3)

############################################
    ddφ,ddθ,ddψ = np.linalg.solve(A,B).T
############################################
    T.my_t=t

    if t>T.progression[0]:
        T.progression.pop(0)
        print('Physical time: ', t, ' Iteration wall clock time: ', datetime.now() - T.wall_clock )
        T.wall_clock = datetime.now()

    return np.concatenate([dx, ddx, dy, ddy, dz, ddz, dφ, ddφ, dθ, ddθ, dψ, ddψ], axis=0)
```

f) Run the solver:

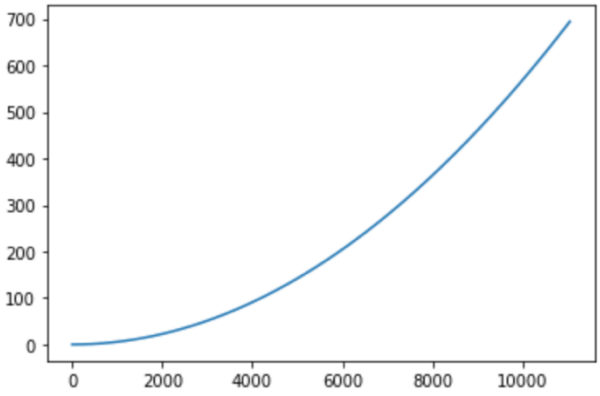
```
tspan = (0., 100)
T_ = MyTime()
```

```
coefs = [1, 1, 1, 1, 1, 1]
```

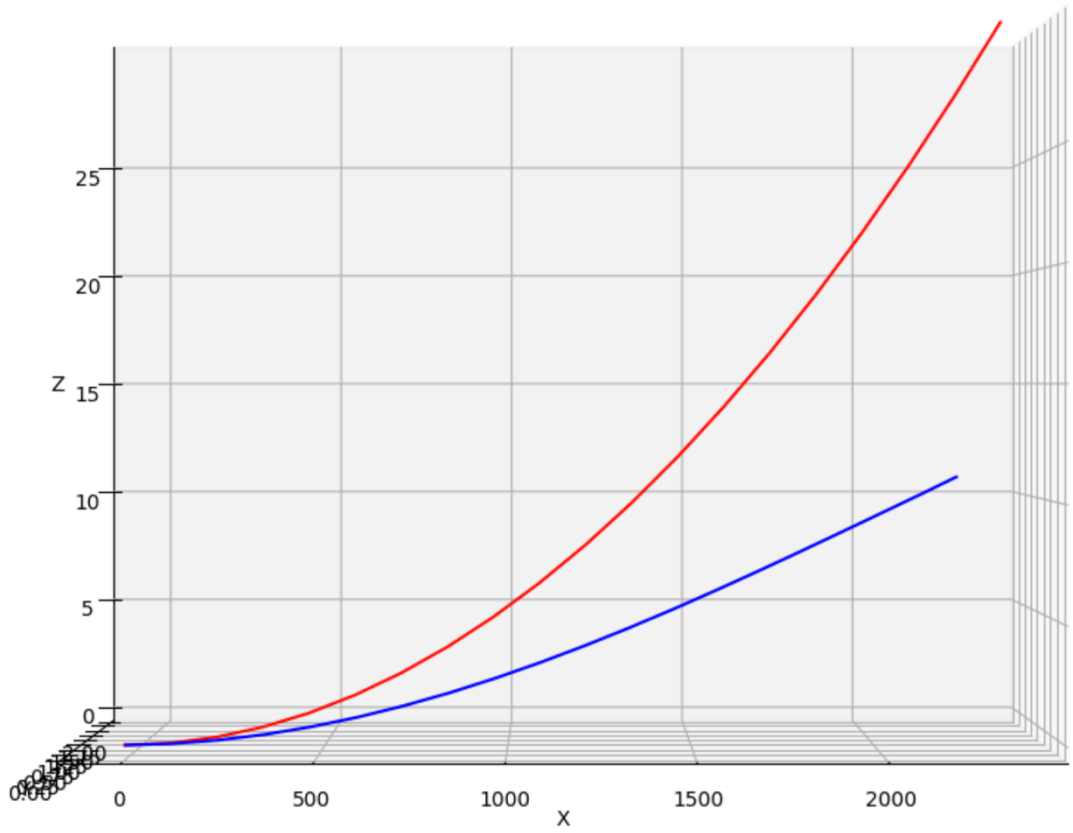
```
startTime1 = datetime.now()
us=solve_ivp(grayscott1d,
             tspan,
             q0,
             args=(coefs,T_,))
print(datetime.now() - startTime1)
```

1.4. Final words

Static solution results:

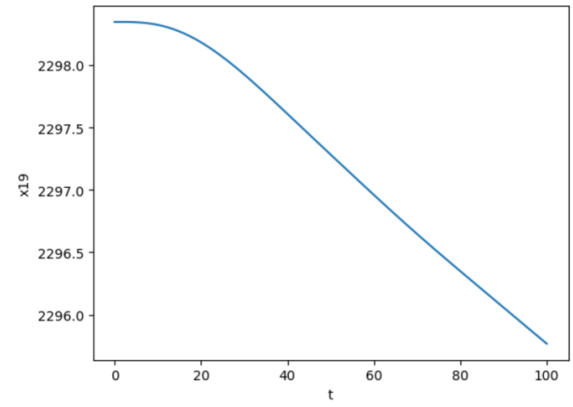


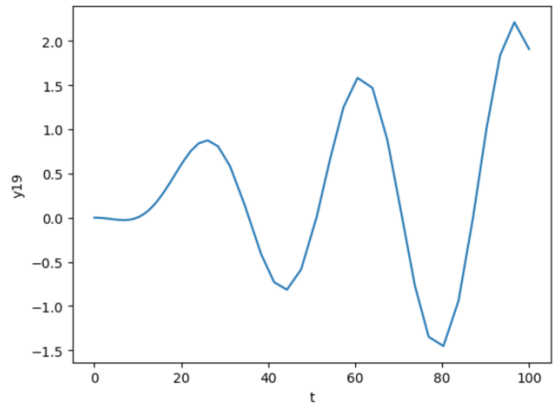
Dynamic solution results(profile):

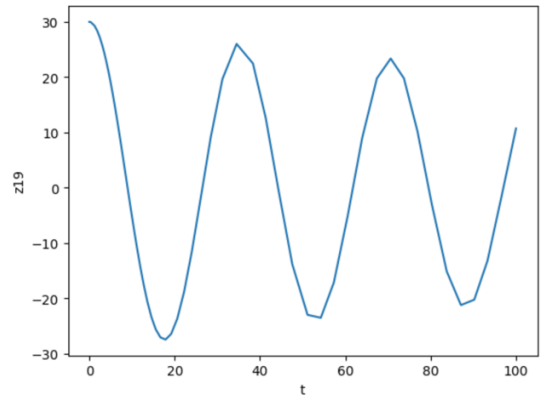


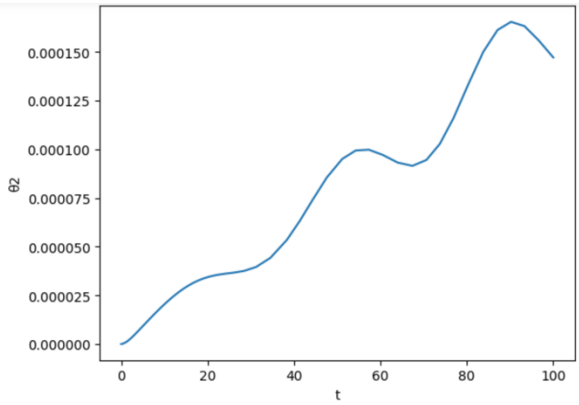
The produced model needs validation on actual data (or other industry recognized software). It produces plausible results in a qualitative sense, but actual numbers are off.

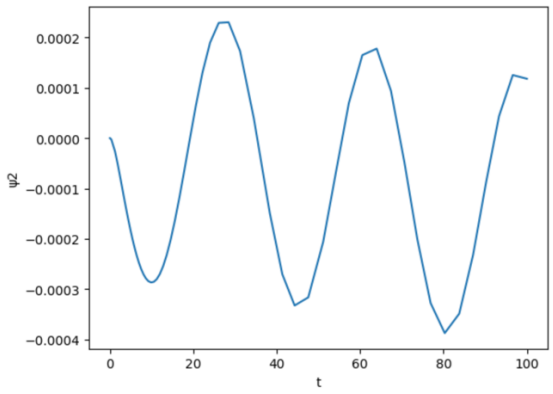
Some dynamic solution results (in time):











1.5. Bibliography

[1] Offshore Pipelaying Dynamics. Gullik Anthon Jensen. Doctoral theses at NTNU, 2010

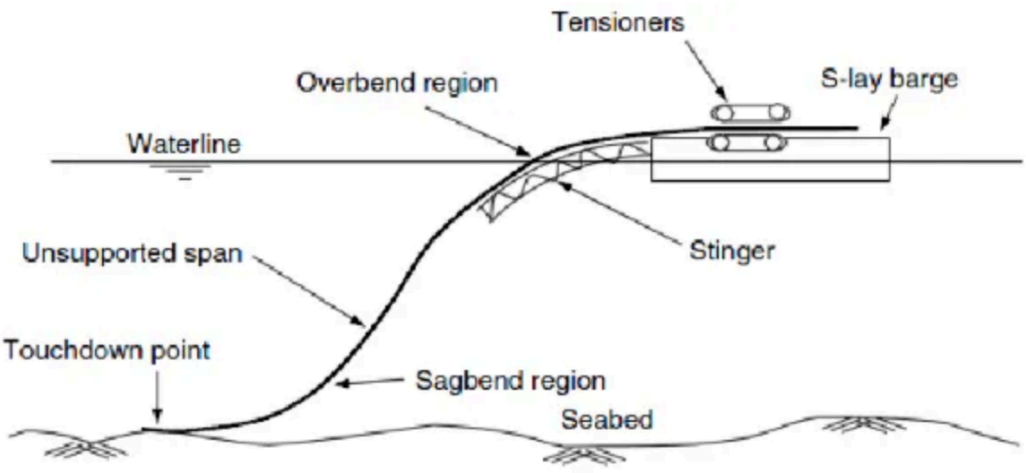
[2] MATLAB Guide to Finite Elements. An Interactive Approach, Peter I. Kattan, 2nd edition

[3] https://github.com/cybergalactic/PythonVehicleSimulator

2. Pipeline laying profile optimization using genetic algorithms

2.1 Background and rationale

The upper curved part of the pipeline is known as the *overbend*. The pipeline will lose contacts with the stinger at a chosen angle and goes downward straightly and then gradually bends in the opposite direction known as the *sagbend* area (hence, the "S" curve). From the sagbend area, the suspended pipe continues to reach the seabed at the touchdown point. The detail of the S-lay configuration is shown below. In the sagbend area, the combination of bending and pressure loads must safely be sustained, [1].



The tension applied at the top is used to control the curvature in the sagbend region. Excessive bending, local buckling and collapse could happen if the tension in the top is lost due to sudden movements of the ship or any other reasons. The main function of the lay vessel is to provide tension to hold the suspended line pipes and to control its shape. The behavior of the long-suspended pipeline is more like a cable rather than a beam (so-called catenary curve), [1].

Usually, engineers try to minimize the top tension to reduce operational costs while preserving tensioners' ability to hold the pipe during installation in rough seas.

In this paper, I propose using genetic algorithm for the vessel laying profile design optimization for successfully conducting subsea pipeline installation procedure. Basically, it is a problem of continuous function optimization with some constraints, for which genetic algorithms are nicely suited. Formally, I aim to minimize top tension of tensioners with the constraints that sagbend tension is no greater than 60% of pipe's Yield Strength (SMYS) and overbend tension is no greater that 90% of SMYS. The function to be minimized is that of analytical (in contrast to numerical, usually done by specialized software) static pipelay analysis with the following assumptions: the lift-off point is assumed to occur at the second from the last roller at the bottom of the stinger. This is where the catenary curve is deemed to join. In fact, it is recognized that the true point of contra-flexion is beyond the end of the stinger; All current and wave forces on the pipeline and barge are ignored; Water absorption is taken as zero because of the short time that the pipe is immersed prior to touchdown; The end cap pressure can thus be taken as acting over the outer diameter of the concrete. For the optimization function derivation, one should contemplate simple considerations of the geometry of the barge, the pipe's friction on the stinger and catenary equation, that can be found, for example, in [1], and is beyond the scope of this paper. Alternatively, more complex optimization function can be used including that of provided by commercial software for static and dynamic analysis of pipelay. In this article, however, I wanted to keep things simple.

2.2. Methodology

2.2.1. Catenary

In physics and geometry, a catenary is the curve that an idealized hanging chain or cable assumes under its own weight when supported only at its ends in a uniform gravitational field. It has well established equation and was used to estimated tension in the pipeline.

2.2.2. Genetic algorithms

According to Wikipedia, in computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimisation and search problems by relying on biologically inspired operators such as mutation, crossover and selection. In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimisation problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible (with e.g., continuous variables like in this paper). The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective

function in the optimisation problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A good introductory book for GAs, extensively used in this article, is [2]. It uses python as a primary programming language and DEAP framework for implementing genetic algorithms. DEAP is a novel evolutionary computation framework for rapid prototyping and testing of ideas. It seeks to make algorithms explicit and data structures transparent. It works in perfect harmony with parallelisation mechanism such as multiprocessing and SCOOP, [3].

2.2.3. Lay profile optimization

GA was used to optimized pipeline lay profile for static analysis.

2.3. Model and implementation

The following steps describe the main parts of the program:

a) Input data

ODs = 323.9	Outer diameter of steel pipe, [mm]
ts = 14.2	Wall thickness of steel pipe, [mm]
Es = 207	Young's modulus of steel, [GPa]
SMYS = 358	SMYS for steel, [MPa]
rho_s = 7850	Density of steel, [kg·m^–3]
tFBE = 0.5	Thickness of FBE insulation layer, [mm]
rhoFBE = 1300	Density of FBE, [kg·m^–3]
tconc = 50	Thickness of concrete coating, [mm]
rho_conc = 2250	Density of concrete, [kg·m^–3]

Table 1: Pipe data

d = 50	*Water depth, [m]*
rho_sea = 1025	*Density of seawater,[kg·m^–3]*

Table 2: Environmental data

mu_roller = 0.1	*Roller friction for pipe on stinger.*

Table 3: Pipe launch rollers

thetaPT = 0…5	Angle of inclination of firing line from horizontal, [deg]
LFL = 80…150	Length of pipe on inclined firing line, [m]
RB = 100…250	Radius of over-bend curve between stinger and straight section of firing line, [m]
ELPT = 5…15	Height of Point of Tangent above Reference Point (sternpost at keel level), [m]
LPT = 5…20	Horizontal distance between Point of Tangent and Reference Point, [m]
ELWL = 3…20	Elevation of Water Level above Reference Point, [m]
LMP = 2…10	Horizontal distance between Reference Point and Marriage Point, [m]
RS = 100…250	Stinger radius, [m]
CL = 40…80	Chord length of the stinger between the Marriage Point and the Lift-Off Point at the second from last roller, [m]

Table 4: Lay-barge data to be optimised with bounds (see Fig.3)

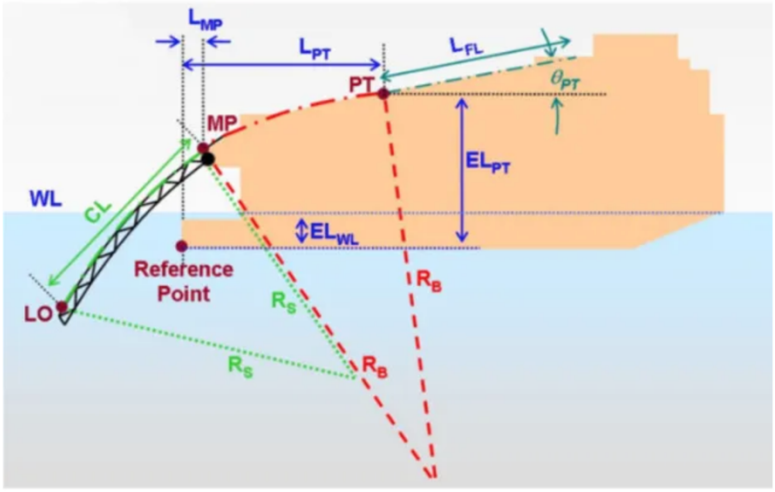


Figure 3: Typical lay-barge schematic.

b) Start by setting the lower and upper boundary for each of the float values representing a Lay-barge data:

```
BOUNDS_LOW = [0, 80, 100, 5, 5, 3, 2, 100, 40]
BOUNDS_HIGH = [5, 150, 250, 15, 20, 20, 10, 250, 80]
```

c) Since our goal is to minimize the top tension of tensioners, we define a single objective, minimization fitness strategy:

```
creator.create(“FitnessMin”, base.Fitness, weights=(-1.0,))
```

d) Now comes a particularly interesting part: since the solution is represented by a list of float values, each of a different range, we use the following loop to iterate over all pairs of lower-bound, upper-bound values. For each entry in Lay-barge data (table 4), we create a separate toolbox operator, which will be used to generate random float values in the appropriate range:

```
for i in range(NUM_OF_PARAMS):
   toolbox.register(“hyperparameter_” + str(i), random.uniform,
BOUNDS_LOW[i],BOUNDS_HIGH[i])
```

e) Then, we create the parameter tuple, which contains the separate float number generators we just created for each parameter:

```
for i in range(NUM_OF_PARAMS):
    hyperparameters = hyperparameters
(toolbox.__getattribute__(“hyperparameter_” + str(i)),)
```

f) Now, we can use this parameter tuple, in conjunction with DEAP's built-in initCycle() operator, to create a new individualCreator operator that fills up an individual instance with a combination of randomly generated parameter values:

```
toolbox.register("individualCreator", tools.initCycle,
 creator.Individual, hyperparameters, n=1)
```

g) Then, we instruct the genetic algorithm to use the pipelayStatic() method instance for fitness evaluation:

```
def piplayStatic(individual):
  # Pipe data
  …….

  # Environmental data
  …….

  # Pipe Launch Rollers
  ……

  # Lay-Barge Input Data
  thetaPT = individual[0]
  LFL = individual[1]
  RB = individual[2]
  ELPT = individual[3]
  LPT = individual[4]
  ELWL = individual[5]
  LMP = individual[6]
  RS = individual[7]
  CL = individual[8]


  Ttens_tonnef, TTS_ratio, TopS_ratio = static_pipe_lay(ODs, ts, Es,
SMYS, rho_s, tFBE, rhoFBE, tconc, rho_conc, d, rho_sea, mu_roller,
thetaPT, LFL, RB, ELPT, LPT, ELWL, LMP, RS, CL)

 if numpy.isnan(Ttens_tonnef):
     Ttens_tonnef = 100

 if TTS_ratio > 0.6 or TTS_ratio < 0.3 or TopS_ratio > 0.9:
     return Ttens_tonnef*PENALTY_VALUE,

 return Ttens_tonnef,

toolbox.register("evaluate", piplayStatic)
```

Here, we use static_pipe_lay() as function for our "black-box" static pipeline installation analysis. We also apply PENALTY_VALUE = 10 if our constraints are violated.

h) Now, we need to define the genetic operators. While, for the selection operator, we use the usual tournament selection with a tournament size of 2, we choose crossover and mutation operators that are specialized for bounded float-list chromosomes and provide them with the boundaries we defined for each parameter:

```
toolbox.register("select", tools.selTournament, tournsize=2)
 toolbox.register("mate",
 tools.cxSimulatedBinaryBounded,
 low=BOUNDS_LOW,
 up=BOUNDS_HIGH,
 eta=CROWDING_FACTOR)

toolbox.register("mutate",
 tools.mutPolynomialBounded,
 low=BOUNDS_LOW,
 up=BOUNDS_HIGH,
 eta=CROWDING_FACTOR,
 indpb=1.0 / NUM_OF_PARAMS)
```

i) In addition, we use the elitist approach, where the HOF (hall-of-fame) members — the current best individuals — are always passed untouched to the next generation:

```
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

population, logbook = elitism.eaSimpleWithElitism(population,
 toolbox,
 cxpb=P_CROSSOVER,
 mutpb=P_MUTATION,
 ngen=MAX_GENERATIONS,
 stats=stats,
 halloffame=hof,
 verbose=True)
```

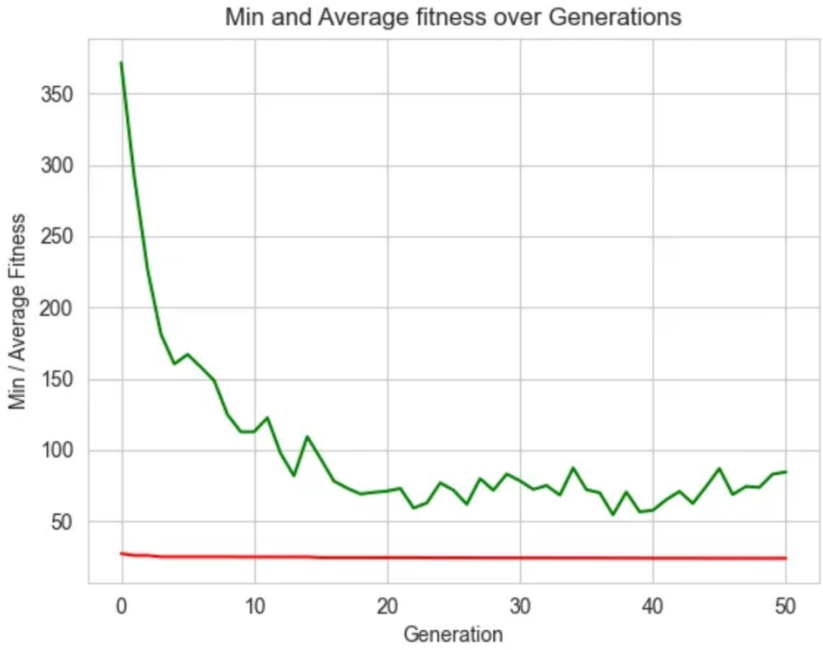
2.4. Final words

By running the algorithm for 50 generations with a population size of 250, we get the following outcome:

— Best Individual = [0.0412993085185044, 81.01473661569246, 184.95018480338263, 9.293609161332556, 14.465039211270776, 9.235358866413463, 7.998131384548541, 116.590780891431, 62.13405099204592]
— Best Fitness = 23.800805434444403

Double check:
Ttens_tonnef: 23.800805434444403
TTS_ratio: 0.5998947240981475 < 0.6
TopS_ratio: 0.8462065818374703 < 0.9

The progress during optimization can be seen below. It is depicting the min (in red) and average (in green) fitness over the generations, indicates that the best solution was approach in less than 5 generations and then only gradually decreased.



2.5. Bibliography

[1] Comparisons Study of S-Lay and J-Lay Methods for Pipeline Installation in Ultra Deep Water. Master's thesis by Jihan Herdiyanti

[2] Hands-On Genetic Algorithms with Python: Applying genetic algorithms to solve real-world deep learning and artificial intelligence problems. Wirsansky, Eyal.

[3] https://deap.readthedocs.io/en/master/

3. Robotic pipe

3.1. Background and rationale

In this article, I am trying to recreate the results of pipelay dynamics obtained in [1, 2] for robotic pipe.

In [1] a model for the pipe is developed as part of a system joining a pipe and a vessel, that is suited for controller tasks while keeping the geometric configuration and the force balance of the pipeline. Instead of discretisation of catenary equations or finite element models which are disadvantageous due to their complexity, the standard robot model for a robot manipulator found in e.g. Spong and Vidyasagar [1989] or Sciavicco and Siciliano [2001] is utilised to model the pipeline. The vessel is included in the pipe model as the last link of the structure. A linked structure with many joints is termed hyper-redundant by Chirikjian and Burdick [1994]. Using a standard robot model formulation is advantageous since tools, developed for robot manipulators, for controller synthesis and stability analysis now can be applied directly.

3.2. Methodology

3.2.1. Pipe model and dynamics

The pipeline is modelled as a series of connected links as illustrated below. It moves in six degrees of freedom (DOF), and hence each pipe element must support longitudinal stretching, lateral bending, and longitudinal rotation. Thus, each element has two rotational and a translational joint. In this paper and in [1] only planar motion of the pipeline in a vertical plane is

considered. Hence the rotation is ignored, and the longitudinal bending is assumed to be small compared to the lateral bending, so it is also ignored. The model is based on the robot equation with minimal coordinates.

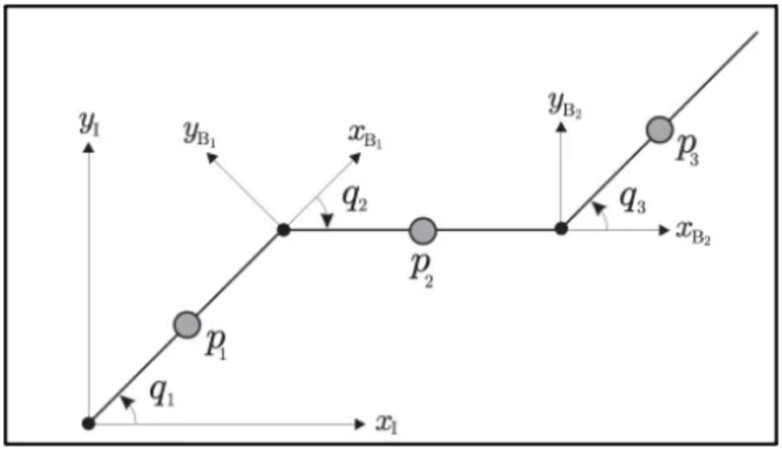


Fig. 2. The three first elements of the slender structure connected by rotational joints, [1]

The dynamics of the system is described by the second order ordinary differential equation (1):

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{H}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{f}(\mathbf{q}) + \mathbf{g}(\mathbf{q}) = \tau$$

Equation (1)

Vector of control inputs consists of the thruster forces and wave forces. Applied top tension is 14k ton-force for water depth of 700m to make the illustration vivid. For more details for the mathematical model and constants used in this article please consult [1,2].

3.2.2. Numerical

The modelling was conducted using SciPy package function *odeint*.

3.3. Model and implementation

The pipe at one end was fixed to sea floor and the other end was attached to center of gravity (CoG) of the vessel.

The model consisted of 4 nodes and depicted in fig.3. Duration of simulation was 600 sec with dt=0.5 sec.

a) For static solution, the robotic model dynamics (1) is reduced to

```
In [82]: F_=np.array([-Fx(q,alp_,dq,kp, 0, x_ref = 0),0]).reshape(2,1)

In [84]: def static_func(q):
             alp=alpha(q)
             ans=  f(q, K) + g(q,alp) - tau(q,alp, tau_qn, F_, 0)
             return ans.reshape((len(q),))

In [85]: root = fsolve(static_func, np.array(q))
```

where the configuration of the pipe is shaped only by the bending stiffness, gravity, and buoyancy (control forces from the stinger is not accounted in the model).

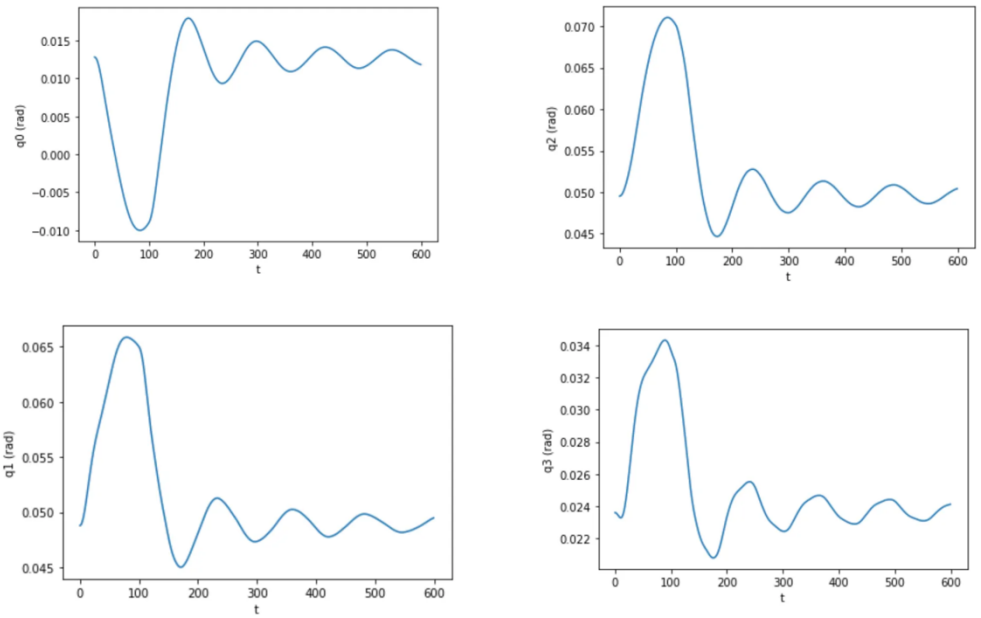
b) After solving statics, the dynamical solution was obtained by applying the main function for the pipe 'manipulator' as presented below:

```
In [102]: def manipulator(Q, t, q):
              q0,dq0,q1,dq1,q2,dq2,q3,dq3 = Q
              QI=np.array([q0,q1,q2,q3]).reshape(len(q),1)
              dQ=np.array([dq0,dq1,dq2,dq3]).reshape(len(q),1)

              QI= QI.flatten().tolist()

              alp=alpha(QI)

              if t>100:
                  F=np.array([-Fx(QI, alp, dQ, kp, kd, x_ref = 0),0]).reshape(2,1)
              else:
                  F=np.array([Fx_,0]).reshape(2,1)


              ddq0,ddq1,ddq2,ddq3 = np.linalg.lstsq(M(QI, l, mn, In, mi),
                                    - np.dot(H(alp,QI,dQ,l), np.radians(dQ)).reshape(len(QI),1)
                                    - np.dot(C(QI,dQ, l, mn, In, mi), np.radians(dQ)).reshape(len(QI),1)
                                    - f(QI, K)
                                    - g(QI,alp)
                                    + tau(QI,alp, tau_qn, F, tau_q_wave), rcond=None)[0]

              return [dq0, ddq0, dq1, ddq1, dq2, ddq2, dq3, ddq3]
```

```
us=odeint(manipulator,q0,t, args=(q,))
```

3.4. Final words

The following diagrams depict coordinates alternations and horizontal tension during pipelay dynamics simulation.



The overall results are mainly along the lines of those obtained in [1,2], considering slight differences in model settings. The dynamics of the pipeline is harsh in terms of variation of the angle vector **q** accounting specific constants used during modelling to make PD controller application more pronounced. Although, we consider the outcomes of this paper as preliminary and requiring verification, the bulk framework is established and can be used in real projects.

3.5. Bibliography

[1] Modelling and Control of Offshore Marine Pipeline during Pipelay Gullik A. Jensen et al, 2008

[2] Offshore Pipelaying Dynamics. Gullik Anthon Jensen. Doctoral theses at NTNU, 2010