AZERBAIJAN REPUBLIC MINISTRY OF EDUCATION

Baku State University
Faculty of Applied Mathematics and Cybernetics
Department of Applied Mathematics

Musa A. Maharramov, Ph.D.
email: musa 'AT' maharramov.com

MATHEMATICAL MODELLING OF A STEADY-STATE FLOW OF A VISCOUS LIQUID IN A PIPELINE

A mathematical model and software for performing PC-based pipeline hydraulics simulations.

Baku 2003

Table of Contents

Introduction

In this work, a mathematical model is presented of a steady-state flow of a viscous liquid-phase hydrocarbon in a steel pipeline. The model has been used in developing computer software and successfully tested in industrial environment.

Chapter I presents the mathematical foundation of the theory developed in the subsequent parts of the work and contains all the principal governing equations.

Chapter II formally introduces the model and adapts the governing equations to its specifics.

Chapter III describes computer software that implements the proposed model.

Appendices A and B contain detailed source code of the application.

Chapter I. Mechanics of Flow in Hydrocarbon Pipelines

This Chapter presents governing equations used in subsequent parts of the work for developing mathematical model of a steady-state flow. The principal objective of this work being the development of computer software for simulating flow of liquid hydrocarbons through steel pipelines, material presented in this Chapter primarily concerns the flow of a viscous incompressible liquid. However, some of the developed apparatus applies to other media as well – e.g., compressible viscous hydrocarbon gas.

1. Governing Equations

Let us consider the flow of a continuous medium (continuum) in the 3-dimensional Euclidean space. It is assumed that the initial position of all particles of the medium is known and the process is described by specifying a velocity field $v^i = v^i\left(x^1, x^2, x^3, t\right), i = 1,2,3$ where $\mathbf{v} = \left(v^1, v^2, v^3\right)$ is the velocity of the particle that has Euler coordinates x^1, x^2, x^3 at the moment of time *t* (i.e., the particle occupying the geometrical point with coordinates x^1, x^2, x^3 at the moment *t*.) It is further assumed that the continuum is characterised by a spatial density $\rho = \rho\left(x^1, x^2, x^3, t\right)$ and internal energy $U = U\left(x^1, x^2, x^3, t\right)$. The last quantity is characteristic of the **total energy** of molecules in a **unit mass** of the medium – i.e., the internal energy is the sum of the total kinetic energy of the molecules and potential energy of their interaction. Note that in a medium where the potential energy of molecule interaction can be neglected (e.g., in the perfect gas) the internal energy is equal to the total kinetic energy of molecules. Another parameter of the medium is temperature which is proportional to the **mean kinetic energy** of the molecules. In practice the inner energy can be a function of temperature and density and/or other parameters that will be described later in this Chapter, however, a particular functional dependence is determined by the medium in question. For instance,

$$U(T, \rho) = \int_{T_0}^{T} c_V(T)\,dT - a\rho + const$$

for a Van der Vaalse gas and

$$U(T, \rho) = U(T) = \int_{T_0}^{T} c_V(T)\,dT + U(T_0) \qquad (1)$$

for the perfect gas and incompressible fluid, wherein $c_V(T)$ is the thermal capacity/specific heat of the medium (at constant density if the internal energy **does** depend on density). Throughout this work we will use formula (1) that establishes relationship between the internal energy and temperature.

Let us denote via $V(t)$ the domain occupied by an infinitesimally small particle of the continuum at a moment t. So long as there occurs no external mass transfer, the particle mass is conserved and the following conservation of mass equation holds:

$$\frac{dm}{dt}=\frac{d}{dt}\int_{V(t)}\rho(x^1,x^3,x^3,)dx=0 \text{ (in Euler coordinates)}$$

or

$$\frac{dm}{dt}=\frac{d}{dt}\int_{V(t=t^*)}\rho\left(x^1\left(\xi^1,\xi^2,\xi^3,t\right),x^2\left(\xi^1,\xi^2,\xi^3,t\right),x^3\left(\xi^1,\xi^2,\xi^3,t\right),t\right)\det\left[\frac{\partial x^i\left(\xi^1,\xi^2,\xi^3,t\right)}{\partial\xi^j}\right]d\xi=0$$

where Euler coordinates at an arbitrary fixed moment $t=t^*$ are used as Lagrange coordinates. Differentiating the integral expression we obtain the following continuity equation:

$$\frac{\partial\rho\left(x^1,x^2,x^3,t\right)}{\partial t}+\operatorname{div}(\rho\mathbf{v})=\rho_t+\nabla_i\left[\rho v^i\right]=0\,.$$

(2)

Any two adjacent infinitesimal volumes of a continuum act upon each other, and the force of their interaction is a surface force proportional to the contact surface between them. If the surface force acting opposite to the direction of (not necessarily parallel to) the unit normal **n** to a surface is denoted $\mathbf{p}_n$ then the momentum balance equation for the above particle can be spelled out like this:

$$\frac{d}{dt}\int_{V(t)}\rho(x^1,x^3,x^3,t)\mathbf{v}(x^1,x^3,x^3,t)dx=\int_{V(t)}\rho(x^1,x^3,x^3,t)\mathbf{F}(x^1,x^3,x^3,t)dx+\int_{\Sigma(t)=\partial V(t)}\mathbf{p}_n d\sigma$$

.

Where **F** stands for the density of external mass forces (internal mass forces are negligible). It can be easily demonstrated that $\mathbf{p}_n=p_n^i=p^{ij}n_j$ where p^{ij} is the contravariant **stress tensor**. Using Gauss integral formula and equation (2) the above equation can be reduced to the following dirrential form

or in differential form:

$$\rho\frac{dv^i}{dt}=\rho\left[\frac{\partial v^i}{\partial t}+v^j\nabla_j v^i\right]=\rho F^i+\nabla_j p^{ij},\ i=1,2,3 \qquad (3)$$

System of equations (3) is referred to as Euler Equations.

For the total (kinetic plus internal) energy of the particle we have:

$$\frac{d}{dt}\int_{V(t)}\rho\left[U+\frac{v^2}{2}\right]dx=\int_{V(t)}\rho\mathbf{F}\mathbf{v}dx+\int_{\Sigma(t)=\partial V(t)}\mathbf{p}_n\mathbf{v}d\sigma-\int_{\Sigma(t)=\partial V(t)}\mathbf{q}_{heat}\mathbf{n}d\sigma$$

– in the above equation we assume that any change of the total energy of the particle is the effect of **work performed by external mass forces**, **work of internal stress** (e.g., surface forces) and **heat transfer** ($\mathbf{q}_{heat}$ stands for the heat flow vector). Using Gauss integral formula we obtain:

$$\rho\left[\frac{dU}{dt}+\frac{d}{dt}\frac{v^2}{2}\right]=\rho\left[\frac{dU}{dt}+\frac{dv^j}{dt}v_i\right]=\rho\left[\frac{dU}{dt}+\left(\frac{\partial v^i}{\partial t}+v^j\nabla_j v^i\right)v_i\right]=\rho F^i v_i+\nabla_j\left[p^{ij}v_i\right]-\nabla_j q^j_{heat} \qquad (4)$$

By virtue of (3) we get:

$$\rho\frac{dv^j}{dt}v_i=\rho\left[\frac{\partial v^i}{\partial t}+v^j\nabla_j v^i\right]v_i=\rho F^i v_i+v_i\nabla_j p^{ij} \qquad (5)$$

Subtracting (5) from (4) and using (1) we arrive at the following equation for internal energy:

$$\rho\frac{dU}{dt}=\rho c_V(T)\frac{dT}{dt}=\rho c_V(T)\left[\frac{\partial T}{\partial t}+v^j\nabla_j T\right]=p^{ij}\nabla_j v_i-\nabla_j q^j_{heat}. \qquad (6)$$

According to Fourier Law for the heat flow we have $q^i_{heat}=\kappa\nabla^i T$ wherein κ is the thermal conductivity coefficient of the medium. Hence (6) takes the form of a heat equation:

$$\rho c_V(T)\left[\frac{\partial T}{\partial t}+v^j\nabla_j T\right]=p^{ij}\nabla_j v_i+\kappa\nabla_j\nabla^j T\,. \qquad (7)$$

Note that (7) assumes that the medium is insulated from the ambient and no heat transfer occurs across the boundary. Otherwise, near the boundary of the domain where the medium is contacting ambient environment, the heat flow will be the combination of the flow within the medium and flow through the boundary (see Fig. 1)

$$\mathbf{q}_{heat}=-\kappa\,\mathrm{grad}\,T+\kappa_{boundary}\delta_{boundary}\left[T-T_{ambinet}\right]\mathbf{n}_{boundary} \qquad (8)$$

where $T_{ambinet}$ is the ambient temperature outside of the domain boundary, $\mathbf{n}_{boundary}$ is the external unit normal to the boundary, $\kappa_{boundary}$ is the heat conductivity of the boundary (that is the amount of heat transferred per second per Calvin from inside the domain where the flow is taking place, through whatever boundary layer and thence across the boundary into the ambient), $\delta_{boundary}$ is the single-layer potential.

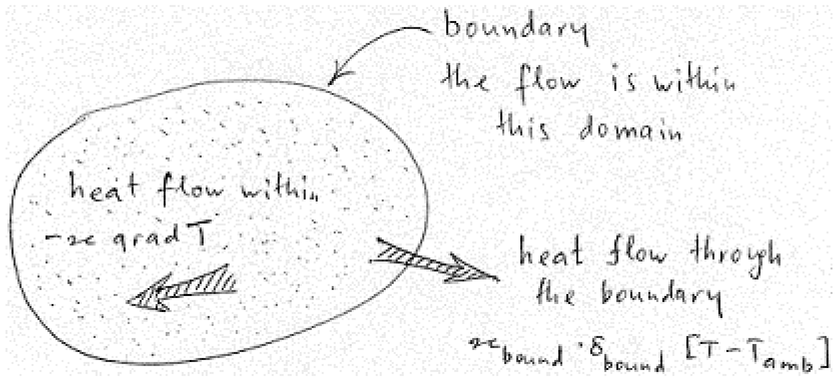

Fig.1. Heat transfer inside the flow and through the boundary

2. Viscous Liquid

In this section we will spell out the governing equations presented in Section 1 for particular cases of viscous incompressible fluid and viscous compressible gas. Furthermore, we will explain the physical meaning of the right-hand side of the equation (6).

In the viscous liquid (gas) the stress tensor yields itself to the following representation:

$$p^{ij} = -pg^{ij} + \tau^{ij}, \ \tau^{ij} = A^{ijkl} e_{kl} \tag{9}$$

where p is a scalar function referred to as pressure, and $e_{kl} = \frac{1}{2}\left(\nabla_l v_k + \nabla_k v_l\right)$ is the deformation rate tensor. If all the components of the contravariant (4,0) tensor A^{ijkl} are 0 then (9) is the stress tensor of the perfect gas/liquid.

If the studied liquid is isotropic (e.g., its properties are invariant of the spatial rotations and reflections) then the contravariant tensor A^{ijkl} can be shown to be determined by two independent parameters and the strain tensor to have the following form:

$$p^{ij} = -pg^{ij} + \lambda g^{ij} \mathbf{div\,v} + 2\mu g^{i\kappa} g^{jl} e_{kl} = -pg^{ij} + \lambda g^{ij} \nabla_k v^k + 2\mu g^{i\kappa} g^{jl} e_{kl} \tag{10}$$

or

$$p^{ij} = -pg^{ij} + \lambda g^{ij} \nabla_k v^k + 2\mu e^{ij} \tag{11}$$

where μ, λ are dynamic viscosity and Lamé coefficient respectively. Substituting (11) into (3) we obtain the following Navier-Stokes equations governing the flow of a viscous compressible liquid:

$$\rho\frac{dv^i}{dt}=\rho\left[\frac{\partial v^i}{\partial t}+v^j\nabla_j v^i\right]=\rho F^i-\mathrm{grad}p+(\lambda+\mu)\,\mathrm{grad}\,\mathrm{div}\,\mathbf{v}+\mu\nabla^k\nabla_k v^i,\ i=1,2,3$$
(12)

that are reduced into

$$\rho\left[\frac{\partial v^i}{\partial t}+v^j\nabla_j v^i\right]=\rho F^i-\mathrm{grad}p+\mu\nabla^k\nabla_k v^i,\ i=1,2,3.$$
(13)

for an incompressible viscous liquid.

The term $(\lambda+\mu)\,\mathrm{grad}\,\mathrm{div}\,\mathbf{v}+\mu\nabla^k\nabla_k v^i,\ i=1,2,3$ in (12, 13) quantifies the effect of internal "friction" between different layers of the liquid flowing at different speeds – hence the derivatives of velocity in (12) – and the term $p^{ij}\nabla_j v_i$ in (6) quantifies the effect of stress on the internal energy. The latter effect can be further elaborated as follows:

$$p^{ij}\nabla_j v_i=-pg^{ij}\nabla_j v_i+\tau^{ij}\nabla_j v_i \qquad (14)$$

and

$$\rho\frac{dU}{dt}=\rho c_V(T)\frac{dT}{dt}=\rho c_V(T)\left[\frac{\partial T}{\partial t}+v^j\nabla_j T\right]=-pg^{ij}\nabla_j v_i+\tau^{ij}\nabla_j v_i-\nabla_j q^j_{heat}$$
(15)

By virtue of (2) we obtain $g^{ij}\nabla_j v_i=\nabla^i v_i=\nabla_i v^i=-\frac{d\rho/dt}{\rho}$, hence $-pg^{ij}\nabla_j v_i=\frac{p}{\rho}\frac{d\rho}{dt}$ and

$$\frac{dU}{dt}=-p\frac{d1/\rho}{dt}+\frac{1}{\rho}\tau^{ij}\nabla_j v_i-\frac{1}{\rho}\nabla_j q^j_{heat} \qquad (16)$$

Note that for a reversible process in a perfect inviscid compressible liquid we would have (see [LS])

$$\frac{dU}{dt}=-p\frac{d1/\rho}{dt}+T\frac{ds}{dt} \qquad (17)$$

where *s* is the entropy. Therefore, assuming that (17) holds for viscous liquids as well (Gibbs Formula), we obtain:

$$T\frac{ds}{dt}=\frac{1}{\rho}\tau^{ij}\nabla_j v_i-\frac{1}{\rho}\nabla_j q^j_{heat} \qquad (18)$$

The term $\frac{1}{\rho}\tau^{ij}\nabla_j v_i$ quantifies the amount of kinetic energy **converted into heat** due to viscosity and (14) is the amount of kinetic energy converted into **internal energy**.

The presented mathematical model fails to take into account the effect of friction between the medium and boundary of the domain where the motion is taking place. Such a friction would result in an irreversible production of heat, contributing a positive term to the right-hand sides of equations (16) and (18), and resulting in an equal decrease of the kinetic and/or potential energy in (12) (Note that the production of heat due to friction does not always result in a corresponding decrease of the kinetic energy – decrease of the potential energy (e.g., pressure) may compensate the loss). The next Chapter presents a mathematical model of the flow of a viscous liquid in a steel pipeline that includes the effect of friction against pipe walls.

Chapter II. Mathematical Model of a Steady-state Flow through a Steel Pipeline

In this Chapter the governing equations introduced in Chapter I are adapted to a mathematical model of the **flow of a viscous incompressible liquid** in a steel pipeline.

3. Pipeline

In this Chapter we develop a mathematical model of the steady-state functioning of a hydrocarbon pipeline. Our ultimate objective is to develop software that will be able to automatically locate PS and PRS along pipeline route and upgrade an existing system of intermediate PS to ensure desired productivity. Unless specified otherwise, the product being transported is considered to be an incompressible viscous liquid (e.g., crude oil, petrol, kerosene, etc). The following is a brief summary of how large hydrocarbon pipeline systems are operated.

A typical large liquid-phase hydrocarbon pipeline transportation system consists of hundreds of kilometres of large diameter (>=400mm) line pipe and auxiliary facilities (valve stations, intermediate pump and pressure reducing stations, metering units, etc.) Due to a relatively high viscosity of hydrocarbon products (especially waxy oils) friction between the transported product and pipeline walls results in sharp pressure drops (see Fig.4). It can be easily seen from the Bernoulli equation $\frac{\rho v^2}{2} - p + gz \equiv const$ that the operating pressure of a steady-state flow cannot drop below the atmospheric pressure. However, in applications the operating pressure is maintained above saturation pressure to prevent product vaporisation because that may result in the formation of gas pockets with potentially unpredictable consequences. Additionally, operational circumstances may require that a specified minimum pressure be delivered at pump or tank suction (see Fig. 7).

The maximum allowed operating pressure of the pipeline is determined based on the yield strength of pipe material, overall diameter and pipe wall thickness (see Fig. 2) using the following formula:

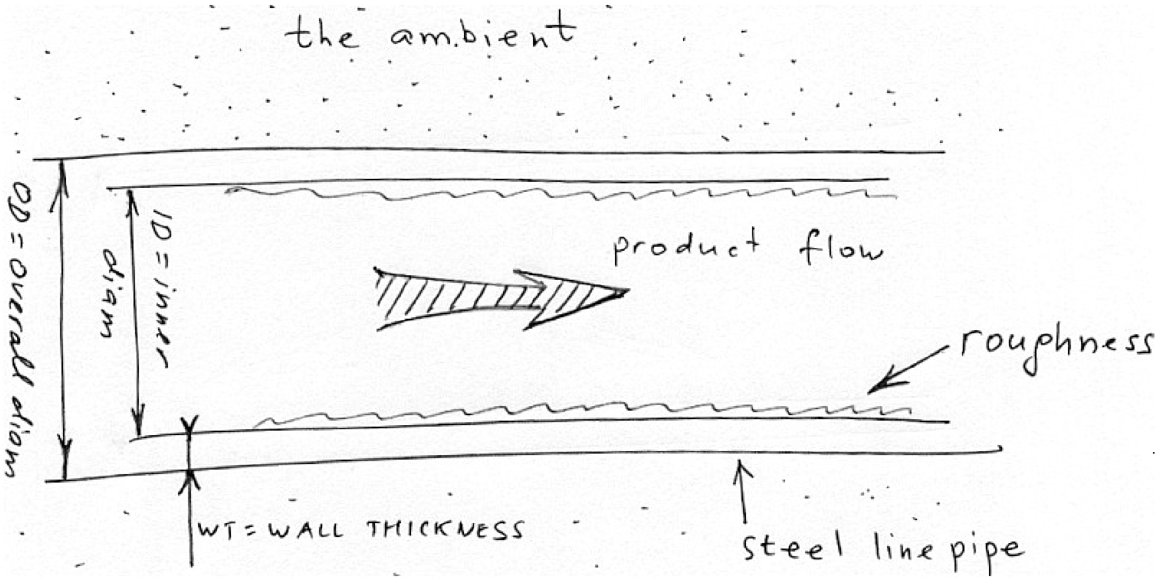

Fig.2. Principal pipeline characteristics affecting the flow.

$$MAOP = 2\frac{WT}{OD} YS \cdot g \cdot DesignFactor \quad (19)$$

where *WT*, *OD*, *YS* denote wall thickness in metres, overall diameter in metres and yield strength in kilograms per square metre. A design factor is applied for safety margin.

MAOP can be converted into the maximum allowed operating head (MAOH) using the formula

$$MAOH = \frac{MAOP}{g\rho} + elevation. \quad (20)$$

MAOP and MAOH are shown as magenta lines on Fig. 7 and Fig. 9 respectively.

The proposed model is based on the following assumptions:

1. velocity of the product is constant at all points of pipeline cross-section and parallel to the pipeline axis;
2. the flow rate is constant;
3. pressure drop (and temperature rise) arises due to friction of product against pipeline walls; the latter depends on the speed of the product, diameter of the line pipe and roughness of the pipe walls;

KM POST	EL (M)	KM (m)	OD (IN)	WT (IN)	YIELD (PSI)	OD (m)	WT (m)	T emp
0	-25.5	0	20.86614	0.314961	47681	0.53	0.008	5
0.126	-11.9	126	20.86614	0.314961	47681	0.53	0.008	5
0.252	-2.4	252	20.86614	0.314961	47681	0.53	0.008	5
0.346	7.56	346	20.86614	0.314961	47681	0.53	0.008	5
0.423	4.48	423	20.86614	0.314961	47681	0.53	0.008	5

0.742	-1.63	742	20.86614	0.314961	47681	0.53	0.008	5
0.935	1.6	935	20.86614	0.314961	47681	0.53	0.008	5
1.078	3.23	1078	20.86614	0.314961	47681	0.53	0.008	5
1.515	14.35	1515	20.86614	0.314961	47681	0.53	0.008	5
1.782	17.28	1782	20.86614	0.314961	47681	0.53	0.008	5
2.384	25.76	2384	20.86614	0.314961	47681	0.53	0.008	5
2.863	20.76	2863	20.86614	0.314961	47681	0.53	0.008	5
3.003	21.5	3003	20.86614	0.314961	47681	0.53	0.008	5
3.362	18.78	3362	20.86614	0.314961	47681	0.53	0.008	5
3.901	16.76	3901	20.86614	0.314961	47681	0.53	0.008	5
3.915	16.41	3915	20.86614	0.314961	47681	0.53	0.008	5
4.281	13.91	4281	20.86614	0.314961	47681	0.53	0.008	5
4.729	12.31	4729	20.86614	0.314961	47681	0.53	0.008	5
4.946	10.98	4946	20.86614	0.314961	47681	0.53	0.008	5
5.361	10.61	5361	20.86614	0.314961	47681	0.53	0.008	5
5.637	8.5	5637	20.86614	0.314961	47681	0.53	0.008	5
5.873	4.64	5873	20.86614	0.314961	47681	0.53	0.008	5
6.199	4.93	6199	20.86614	0.314961	47681	0.53	0.008	5

Fig.3. Fragment of a pipeline datasheet. Columns contain KM posts, elevation, overall line pipe diameter in inches, line pipe wall thickness in inches, steel yield strength in pounds per square inch, ambient temperature in degrees Celsius.

4. any flow and pipeline parameters are assumed constant within discrete segments in between
km points but change across segments (see Fig. 3 and 4);

5. temperature changes slowly along the pipeline and heat transfer through the product can be neglected.

Fig.3. demonstrates a fragment of a typical pipeline datasheet. All parameters are specified for discreet segments each from a few hundred to a few thousand metres in length.

4. Model

The purpose of this section is to adapt the governing equations of Chapter I to the problem in question based on the assumptions of Section 3.

We will assume that the kinematical viscosity and density are known functions depending "slowly" on temperature – i.e., $\nu = \frac{\mu}{\rho} = f(\varepsilon T),\ \varepsilon \ll 1$ and $\rho = g(\varepsilon T),\ \varepsilon \ll 1$, flow rate Q and product specific heat c_V are constants. Given a value of the desired head at terminal, pipeline profile and properties (see Fig. 3), we will locate PS and PRS along the route so as to ensure the desired steady-state flow.

For velocity of the product in segment *i* we have:

$$v(i) = \frac{4Q}{\pi D^2(i)} \tag{21}$$

where $v(i), D(i)$ are speed of the product in, and inner diameter of, the *i*-th pipeline segment.

For the momentum balance (see (3)) we obtain:

$$\rho(T(i))\left[\frac{\partial v(i)}{\partial t}+v(i)\frac{v(i)-v(i-1)}{km(i)-km(i-1)}\right]=$$
$$=\rho(T(i))v(i)\frac{v(i)-v(i-1)}{km(i)-km(i-1)}=-\rho(T(i))g\frac{\mathrm{el}(i)-\mathrm{el}(i-1)}{km(i)-km(i-1)}-$$
$$-\frac{p(i)-p(i-1)}{km(i)-km(i-1)}-\rho(T(i))\frac{\lambda(i)}{2D(i)}v(i)|v(i)|$$

(22)

where el*(i)* is the elevation of the *i*-th segment, $p(i)$ is the product pressure in the segment, $km(i-1)$, $km(i)$ are the beginning and the end of the segment. The last term in the right-hand side of (22) is the semi-empirical Darcy friction term (see [DH]), $\lambda(i)$ is a dimensionless friction factor. We use the following empirical rule for calculating the latter:

If the Reynolds number $\mathrm{Re}=\mathrm{Re}(i)=\dfrac{v(i)D(i)}{\nu(i)}\leq 2000$ **then**

$$\lambda(i)=\frac{64}{\mathrm{Re}(i)} \qquad (23)$$

otherwise

$$\lambda^{-0.5}(i)=-2\cdot\lg\left(\frac{r(i)}{3.7D(i)}+\frac{2.51}{\mathrm{Re}(i)}\lambda^{-0.5}(i)\right)$$

(24)

where $r(i)$ is the pipeline wall roughness (see Fig.2 and 3).

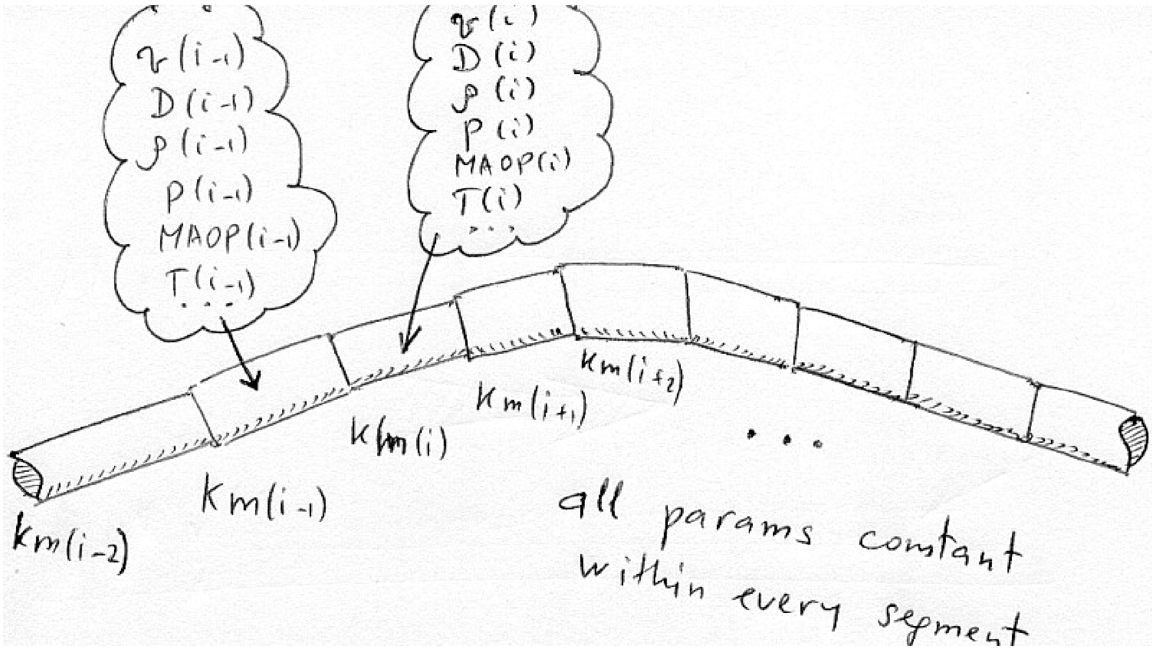

Fig.4. All flow and pipeline parameters are assumed constant within discrete segments in between
km points but may change across segments.

Newton's bi-quadratic method can be used for solving the algebraic equation (24).

Eventually, for energy balance we have:

$$\rho(T(i))c_V\left[\frac{\partial T(i)}{\partial t}+v(i)\frac{T(i)-T(i-1)}{km(i)-km(i-1)}\right]=$$
$$=\rho(T(i))c_V v(i)\frac{T(i)-T(i-1)}{km(i)-km(i-1)}=-\frac{4U(i)}{D(i)}\left[T(i)-T_{amb}(i)\right]+\rho(T(i))\frac{\lambda(i)}{2D(i)}|v(i)|^3$$
(25)

where $U(i)$ denotes the overall heat transfer coefficient through the turbulent boundary layer and pipeline wall, and $T_{amb}(i)$ is the ambient temperature at the *i*-th segment. Note that heat transfer through the product is neglected based on the assumption 5 of the previous Section. Heat conductivity through the turbulent boundary layer is adequately described by the following (empirical) formula:

$$\kappa_{blayer}(i)=\lambda(i)c_V\,\rho(T(i))v(i)/8$$
(26)

Hence for $U(i)$ as the heat transfer coefficient of two adjacent media we obtain:

$$U(i)=\left(\kappa_{blayer}(i)^{-1}+\kappa_{wall}(i)^{-1}\right)^{-1}$$
(27)

where $\kappa_{wall}(i)$ is the heat conductivity of the *i*-th segment's walls.

Chapter III. Flow Simulation

This chapter briefly discusses an application that implements the model described in the previous Chapter (see [MM]).

5. Interfaces and Output

Fig. 5 and 6 show principal data entry and control interfaces of the application. The application has been implemented using VBA modules imbedded in an MS Excel spreadsheet.

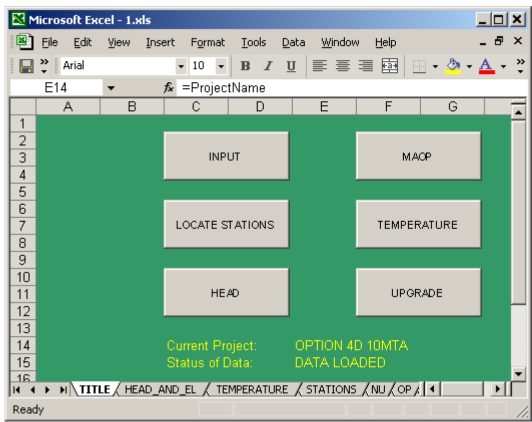

Fig.5. Principal System Interface.

The application's data input interface (see Fig. 6) allows the user to feed all pipeline and product parameters and initial values in a tabulated format in a spreadsheet (se Fig.3).

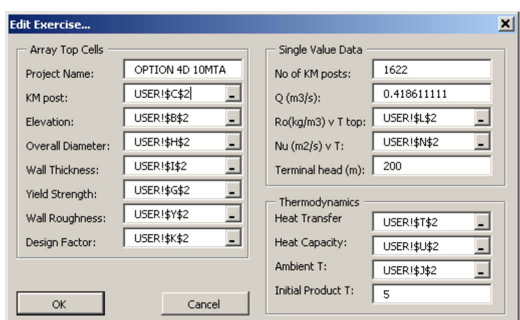

Fig.6. Data Input Interface.

Fig. 7-12 illustrate the application's sample output that is generated using Excel's charting capabilities.

6. Location of PS and PRS

The application's capabilities of automated PS/PRS location as well as upgrade of an existing layout are the features that make this application unique in its class.

The following upgrade was suggested by the application for the pipeline whose profile is shown on Fig.9 which had two existing pump stations at Km 350 and Km 650:

Suggested Upgrade (+1 = PS, -1 = PRS):

972196.7	11	-1
920000		1
789084.9		1
763065.9		-1
696856.8		1
592107.2		1
497139.4		1
390000		1
312108.2		1
206103.4		1
70201.13		1
118597.2		1
16368.82		1

The following listing shows the application’s UpgradeStations() function.

```
Public Sub UpgradeStations()
Dim i As Integer, j As Integer, x As Double, h As Double, Delta As Double
Dim k As Integer, upgrade As Integer
Dim old_h As Double, h1 As Double

    If Not DataLoaded Then MsgBox "No data": Exit Sub
    TerminalHead = Val(Worksheets("DATA").Range("TerminalHead").Value)
    N = Val(Worksheets("DATA").Range("NoOfKmPosts").Value)

    If TerminalHead < Profile(N - 1) Then _
        MsgBox "Destination head must be in excess of elevation", , _
        "Ordos 99": Exit Sub
    ProgressOn
    With Application.Worksheets("DATA")
        UpgradedNoOfStations = .Range("UPGRADENOOFSTATIONS").Cells(1, 1).Value
        For j = 0 To UpgradedNoOfStations - 1
            UpgradeStX(j) = .Range("UPGRADESTATIONSX").Cells(1 + j, 1).Value
            UpgradeStation(j) = .Range("UPGRADESTATIONTYPES").Cells(1 + j, 1).Value
            Call ShowProgress("Loading upgraded station data...", j + 1, UpgradedNoOfStations)
        Next j
    End With

    ProgressOff
    ProgressOn
    NoOfStations = 0: h = TerminalHead: i = N - 1: x = Km(N - 1)
    upgrade = 0
    ' x = latest studied node co-ordinate
    While i > 0
```

```
        i = i - 1
        h_old = h
        h = h + Gradient(i) * (x - Km(i))
        If Km(i) <= UpgradeStX(upgrade) And upgrade < UpgradedNoOfStations Then
            x = UpgradeStX(upgrade)
            h = h - Gradient(i) * (x - Km(i))
            upgrade = upgrade + 1
            If UpgradeStation(upgrade - 1) = 1 Then ' upgraded pump
                Delta = h
                GoTo AddPump
            Else ' upgraded regulator
                GoTo AddRegulator
            End If
        ElseIf h <= Profile(i) Then ' run into ground - regulator site
            x = IntersectionOf(Km(i), Km(i + 1), Profile(i), Profile(i + 1), Km(i), x, h, h_old)

            ' equate h to just elevation of point x on the profile
            h = Profile(i) + (x - Km(i)) * (Profile(i + 1) - Profile(i)) / (Km(i + 1) - Km(i))
            ' add a reduction station
AddRegulator:
            StX(NoOfStations) = x: Station(NoOfStations) = -1

            j = HighPoint(x, h)

            If j = -1 Then MsgBox "Error High Point", , "Ordos 99": Exit Sub

            Delta = 0
            If j < i Then
                For k = j To i - 1
                    Delta = Delta + (Km(k + 1) - Km(k)) * Gradient(k)
                Next k
            End If
            Delta = Delta + (x - Km(i)) * Gradient(i)

            ' profile too high
            'If Profile(El(j)) > Operating(j) Then
            '    MsgBox "Elevation at point " & Format(Km(j), "##.##") & " exceeds OP", , _
            '    "Ordos 99 - calculation aborted"
            '    ProgressOff
            '    Exit Sub
            'End If

            ' head reduction
            DeltaH(NoOfStations) = Profile(j) - Delta - h
            h = h + DeltaH(NoOfStations)
            NoOfStations = NoOfStations + 1
            h = h + 0.01 ' margin
        ElseIf h >= PumpOrOp(i) Then ' exceeded OP - pump site
            x = IntersectionOf(Km(i), Km(i + 1), _
            PumpOrOp(i), PumpOrOp(i + 1), _
                    Km(i), x, h, h_old)

            Delta = PumpOrOp(i) + (x - Km(i)) * _
                  (PumpOrOp(i + 1) - PumpOrOp(i)) / _
                  (Km(i + 1) - Km(i))
AddPump:
            ' minimum suction pressure
            h = Profile(i) + (x - Km(i)) * _
                  (Profile(i + 1) - Profile(i)) / _
                  (Km(i + 1) - Km(i))
            h = Maximum(h, El(i) + MinPumpSuct / (Density(i) * gravity))
            StX(NoOfStations) = x

            j = HighPoint(x, h)

            If j <> -1 Then
                h1 = Profile(j)
                If j < i Then
                    For k = j To i - 1
```

```
                h1 = h1 - (Km(k + 1) - Km(k)) * Gradient(k)
            Next k
        End If
        h1 = h1 - (x - Km(i)) * Gradient(i)
        If h1 > h Then h = h1
    End If

    Delta = Delta - h

    ' add a pump station
    Station(NoOfStations) = 1
    DeltaH(NoOfStations) = Delta
    NoOfStations = NoOfStations + 1
    h = h + 0.01 ' margin
  Else ' carry on OK
    x = Km(i)
  End If
  If i Mod 2 = 0 Then _
    Call ShowProgress("Upgrading station layout...", N - i, N - 1)
Wend
ProgressOff
ProgressOn
' save station locations and delta head
With Application.Worksheets("DATA")
  .Range("NOOFSTATIONS").Cells(1, 1).Value = _
        NoOfStations
  For j = 0 To NoOfStations - 1
    .Range("STATIONS").Cells(1 + j, 1).Value = _
      StX(j)
    .Range("DELTAHEAD").Cells(1 + j, 1).Value = _
      DeltaH(j)
    .Range("STATIONTYPES").Cells(1 + j, 1).Value = _
      Station(j)
  If j Mod 5 = 0 Then _
    Call ShowProgress("Storing station data...", j, N - 1)
  Next j
End With

ProgressOff
Call InstallStations
End Sub
```

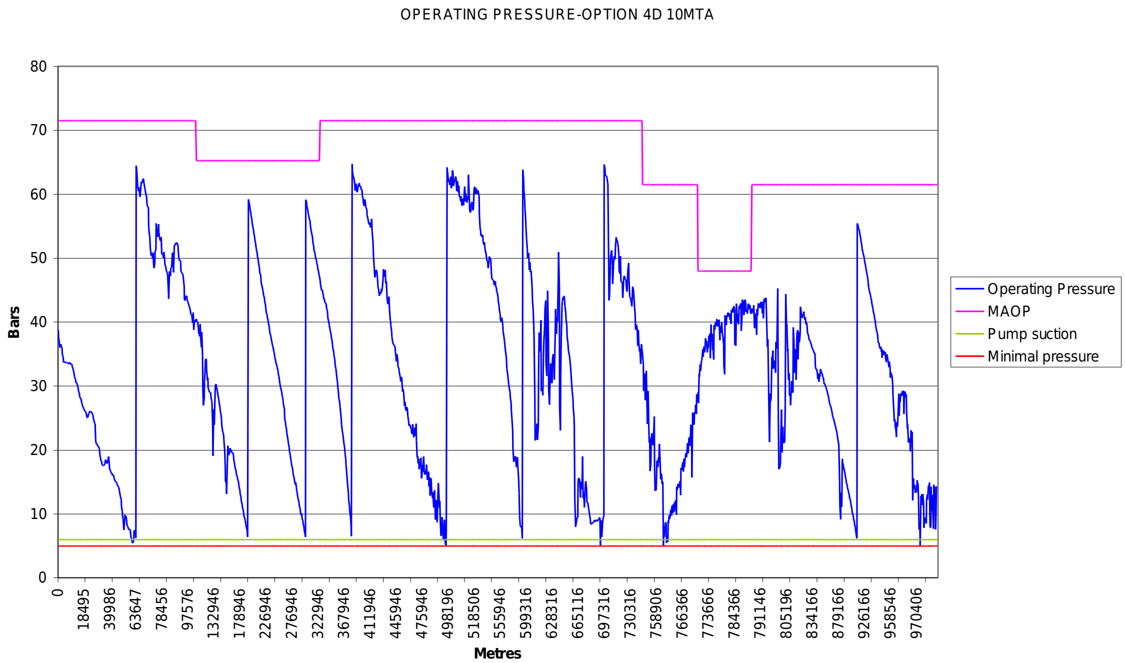

Fig.7. Pipeline MAOP and Operating Pressure. Drop in MAOP at around Km780 is
due to a segment of old pipe.

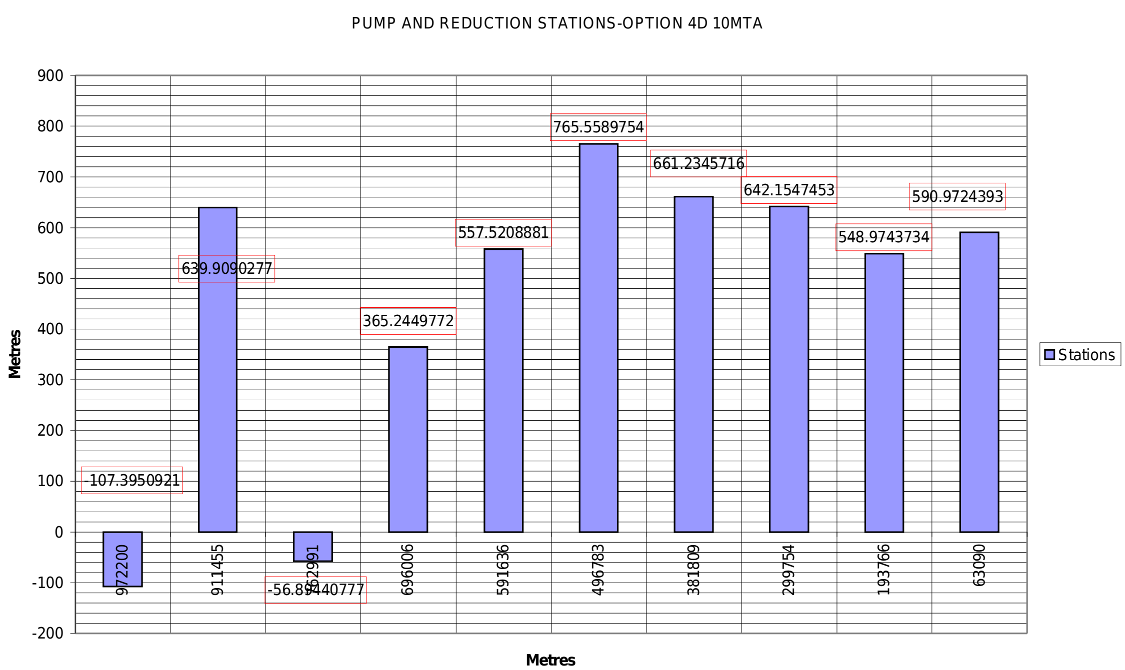

Fig.8. Qty 2 PRS and Qty 8 PS along a 1000 km 10MTA oil pipeline

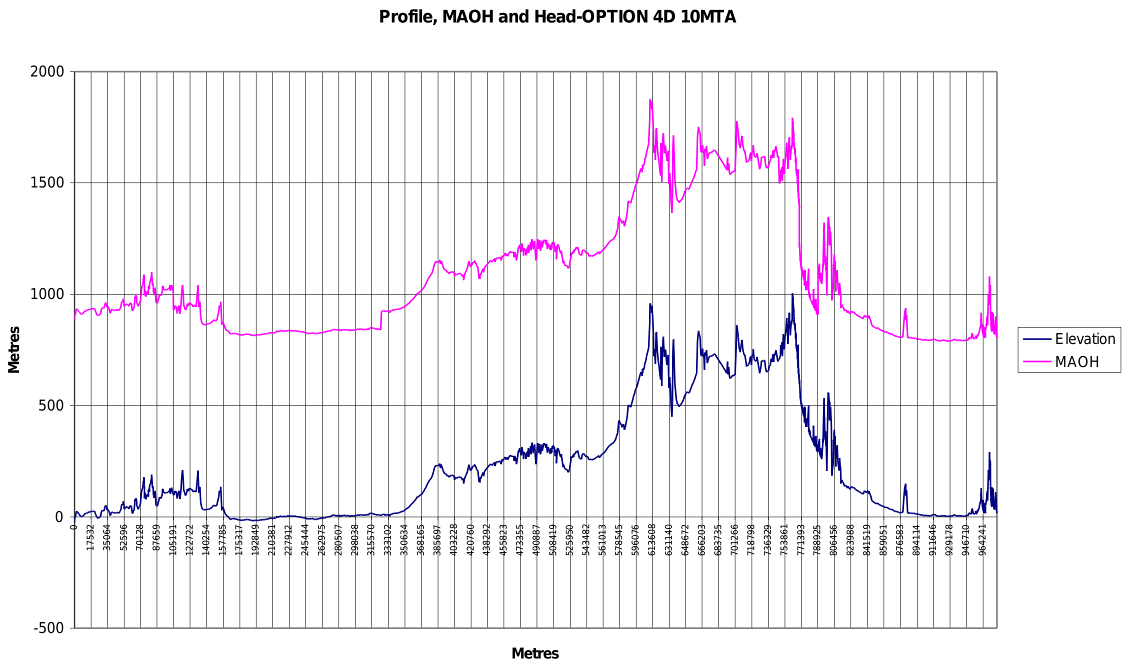

Fig.9. MAOP converted into MAOH (magenta line)

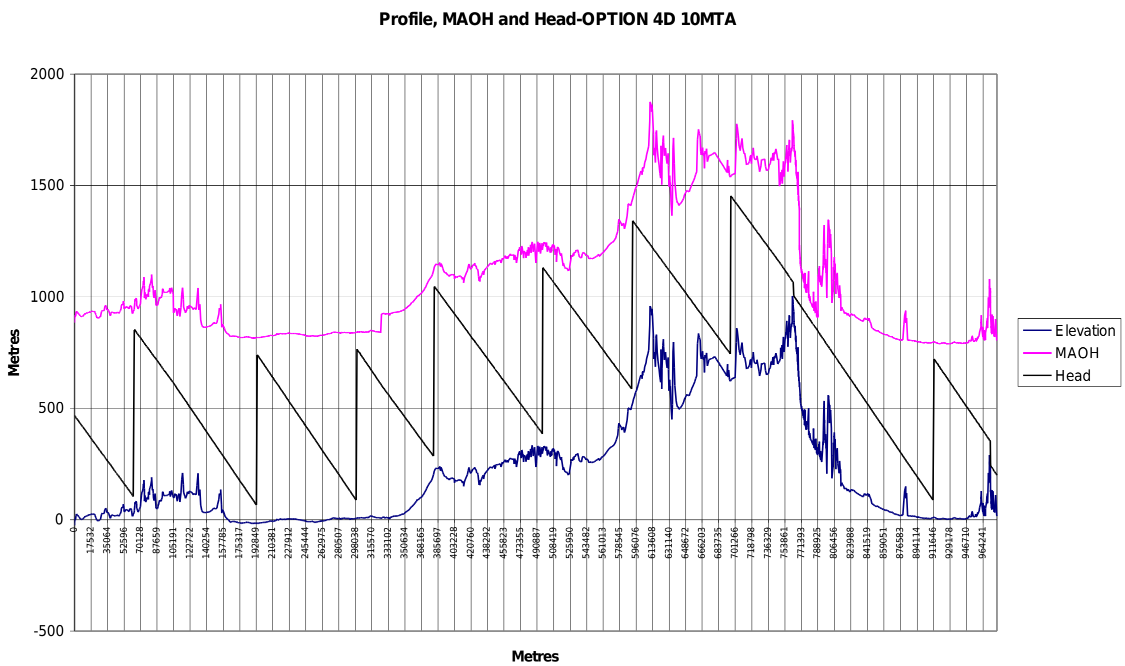

Fig.10. Elevation, head and MAOH. Vertical rises of head match PS locations. Vertical drops are PRS locations. Slanted head indicate pressure/head gradient due to friction.

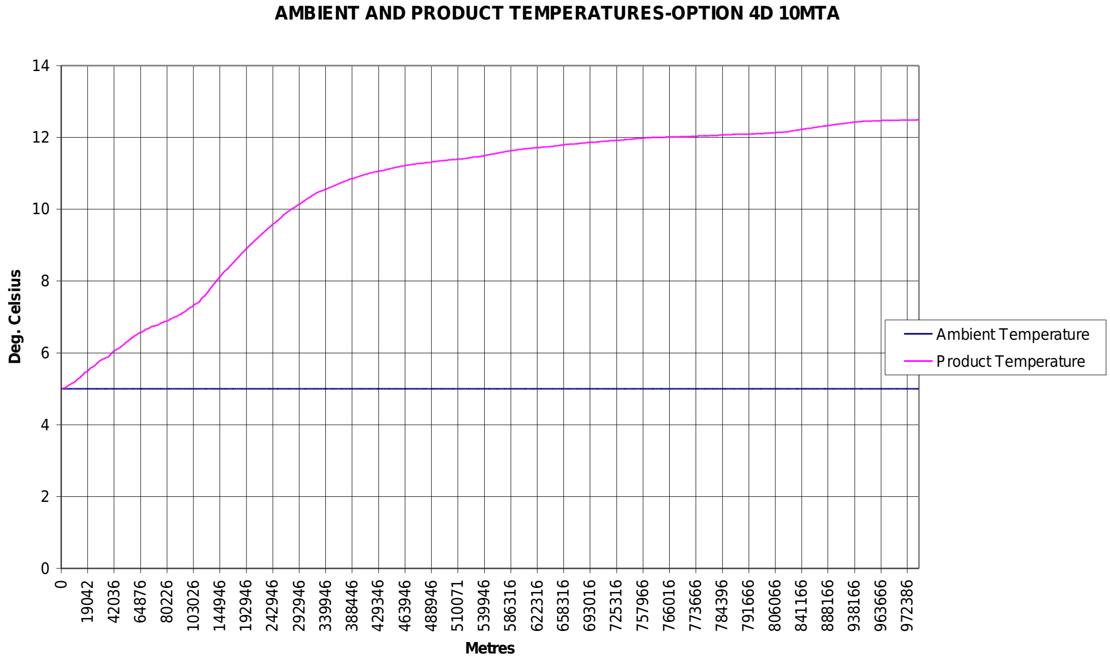

Fig.11. Product and ambient temperature. The product temperature steadily grows from 5 to over 12 degrees Celsius due to irreversible production of heat.

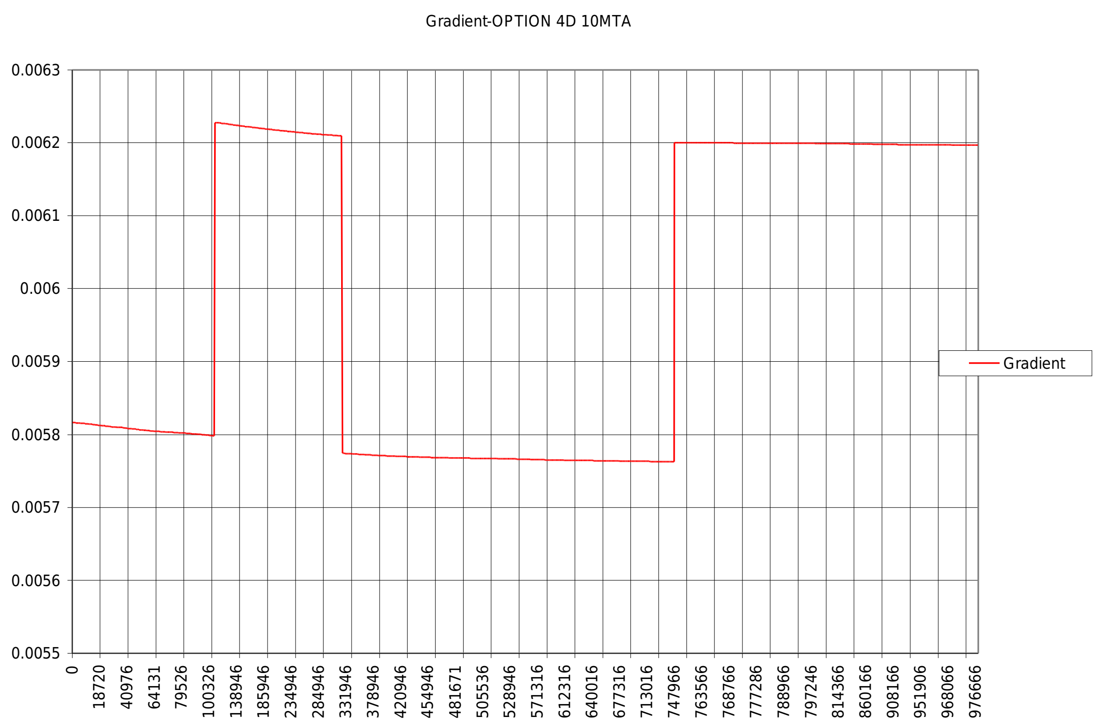

Fig.12. Pressure gradient. Two dips at the beginning and in the centre correspond to a higher quality line pipe.

Appendix A. Source Code of the Data Module

```
Public Sub Edit_Exercise()
    With Application.Worksheets("DATA")
        EditExercise.Controls("ProjectName").Text = _
        .Range("ProjectName").Value

        EditExercise.Controls("NoOfKmPosts").Value = _
        .Range("NoOfKmPosts").Value

        EditExercise.Controls("Q").Value = _
        .Range("Q").Value

        EditExercise.Controls("SGCell").Value = _
        .Range("SGCell").Value

        EditExercise.Controls("TerminalHead").Value = _
        .Range("TerminalHead").Value

        EditExercise.Controls("KMPostCell").Text = _
        .Range("KMPostCell").Value

        EditExercise.Controls("ElevationCell").Text = _
        .Range("ElevationCell").Value

        EditExercise.Controls("ODCell").Text = _
        .Range("ODCell").Value

        EditExercise.Controls("WTCell").Text = _
        .Range("WTCell").Value

        EditExercise.Controls("YSCell").Text = _
        .Range("YSCell").Value

        EditExercise.Controls("RoughnessCell").Text = _
        .Range("RoughnessCell").Value

        EditExercise.Controls("HTRCell").Text = _
        .Range("HTRCell").Value

        EditExercise.Controls("HCCell").Text = _
        .Range("HCCell").Value

        EditExercise.Controls("InitialTemperature").Text = _
        .Range("InitialTemperature").Value

        EditExercise.Controls("DesignFactorCell").Text = _
        .Range("DesignFactorCell").Value

        EditExercise.Controls("TemperatureCell").Value = _
        .Range("TemperatureCell").Value
```

```
        EditExercise.Controls("KVCell").Value = _
        .Range("KVCell").Value

    End With
    VBAProject.EditExercise.Show
End Sub

Public Sub Store_Exercise()
    With Application.Worksheets("DATA")
        .Range("ProjectName").Value = _
        EditExercise.Controls("ProjectName").Text

        .Range("DesignFactorCell").Value = _
        EditExercise.Controls("DesignFactorCell").Text

        .Range("NoOfKmPosts").Value = _
        EditExercise.Controls("NoOfKmPosts").Value

        .Range("Q").Value = _
        EditExercise.Controls("Q").Value

        .Range("TemperatureCell").Value = _
        EditExercise.Controls("TemperatureCell").Value

        .Range("TerminalHead").Value = _
        EditExercise.Controls("TerminalHead").Value

        .Range("KMPostCell").Value = _
        EditExercise.Controls("KMPostCell").Text

        .Range("ElevationCell").Value = _
        EditExercise.Controls("ElevationCell").Text

        .Range("ODCell").Value = _
        EditExercise.Controls("ODCell").Text

        .Range("WTCell").Value = _
        EditExercise.Controls("WTCell").Text

        .Range("YSCell").Value = _
        EditExercise.Controls("YSCell").Text

        .Range("RoughnessCell").Value = _
        EditExercise.Controls("RoughnessCell").Text

        .Range("HTRCell").Value = _
        EditExercise.Controls("HTRCell").Text

        .Range("HCCell").Value = _
        EditExercise.Controls("HCCell").Text

        .Range("InitialTemperature").Value = _
        EditExercise.Controls("InitialTemperature").Text
```

```
        .Range("DesignFactorCell").Value = _
        EditExercise.Controls("DesignFactorCell").Text

        .Range("TemperatureCell").Value = _
        EditExercise.Controls("TemperatureCell").Value

        .Range("KVCell").Value = _
        EditExercise.Controls("KVCell").Value

        .Range("SGCell").Value = _
        EditExercise.Controls("SGCell").Value

    End With
    MATH.LoadData

End Sub



Public Sub ShowProgress(ByVal JobName As String, _
ByVal Value As Double, ByVal MaxValue As Double)
    Application.StatusBar = JobName & " " & _
     Format(Value * 100 / MaxValue, "##") & "% done"
End Sub

Public Sub ProgressOn()
    Application.StatusBar = ""
End Sub

Public Sub ProgressOff()
    Application.StatusBar = "Ready"
End Sub

' return sheet name
Public Function SheetName(ByVal CellName As String) As String
    Dim i As Integer
    i = InStr(1, Range(CellName).Worksheet, "!", 1)
    If i > 0 And i <> Null Then
        SheetName = Left(CellName, i - 1)
    Else
        SheetName = ""
    End If
End Function

Public Function RangePointedToBy(ByVal CellName As String) As String
    Dim SName As String
    Dim r1, c1, r2, c2
    r1 = Range(Application.Range(CellName).Value).Cells(1, 1).Row
    c1 = Range(Application.Range(CellName).Value).Cells(1, 1).Column
    r2 = Range(Application.Range(CellName).Value).Cells(N, 1).Row
    c2 = Range(Application.Range(CellName).Value).Cells(N, 1).Column
    SName = Range(Application.Range(CellName).Value).Worksheet.Name
    RangePointedToBy = SName & "!R" & Format(r1, "#") & "C" & _
        Format(c1, "#") & ":R" & Format(r2, "#") & "C" & Format(c2, "#")
End Function
```

```
Public Function RangeWithTopAt(ByVal CellName As String) As String
    Dim SName As String
    Dim r1, c1, r2, c2
    r1 = Range(CellName).Cells(1, 1).Row
    c1 = Range(CellName).Cells(1, 1).Column
    r2 = Range(CellName).Cells(N, 1).Row
    c2 = Range(CellName).Cells(N, 1).Column
    SName = Range(CellName).Worksheet.Name
    RangeWithTopAt = SName & "!R" & Format(r1, "#") & "C" & _
        Format(c1, "#") & ":R" & Format(r2, "#") & "C" & Format(c2, "#")
End Function

Public Function ColumnWithTopAtOfHeight(ByVal CellName As String, ByVal Height As
Integer) As String
    Dim SName As String
    Dim r1, c1, r2, c2
    r1 = Range(CellName).Cells(1, 1).Row
    c1 = Range(CellName).Cells(1, 1).Column
    r2 = Range(CellName).Cells(Height, 1).Row
    c2 = Range(CellName).Cells(Height, 1).Column
    SName = Range(CellName).Worksheet.Name
    ColumnWithTopAtOfHeight = SName & "!R" & Format(r1, "#") & "C" & _
        Format(c1, "#") & ":R" & Format(r2, "#") & "C" & Format(c2, "#")
End Function



Public Sub ModifyHeadPlot()
    Sheets("HEAD_AND_EL").Select
    With ActiveChart
        .ChartType = xlLine
        .SeriesCollection(1).XValues = "=" & RangePointedToBy("KmPostCell")

        .SeriesCollection(1).Values = "=" & RangePointedToBy("ElevationCell")

        .SeriesCollection(1).Name = "=""Elevation"""
        .SeriesCollection(2).Values = "=" & RangeWithTopAt("MAOH")
        .SeriesCollection(2).Name = "=""MAOH"""
        .SeriesCollection(3).Values = "=" & RangeWithTopAt("Head")
        .SeriesCollection(3).Name = "=""Head"""
        .HasTitle = True
        .ChartTitle.Characters.Text = "Profile, MAOH and Head" & "-" &
Application.Worksheets("DATA").Range("ProjectName").Value
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Metres"
        .Axes(xlValue, xlPrimary).HasTitle = True
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Metres"
    End With
End Sub

Public Sub ModifyGradientPlot()
    Sheets("GRADIENT").Select
    With ActiveChart
        .ChartType = xlLine
```

```
        .SeriesCollection(1).XValues = "=" & RangePointedToBy("KmPostCell")
        .SeriesCollection(1).Values = "=" & RangeWithTopAt("GRAD")
        .SeriesCollection(1).Name = "=""Gradient"""

        .HasTitle = True
        .ChartTitle.Characters.Text = "Gradient" & "-" &
Application.Worksheets("DATA").Range("ProjectName").Value
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Metres"
        .Axes(xlValue, xlPrimary).HasTitle = True
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = ""
    End With
End Sub



Public Sub ModifyTemperaturePlot()
    Sheets("TEMPERATURE").Select
    With ActiveChart
        .ChartType = xlLine
        .SeriesCollection(1).XValues = "=" & RangePointedToBy("KmPostCell")

        .SeriesCollection(1).Values = "=" & RangePointedToBy("TemperatureCell")

        .SeriesCollection(1).Name = "=""Ambient Temperature"""
        .SeriesCollection(2).Values = "=" & RangeWithTopAt("TEMP")
        .SeriesCollection(2).Name = "=""Product Temperature"""
        .HasTitle = True
        .ChartTitle.Characters.Text = "AMBIENT AND PRODUCT TEMPERATURES" & "-" &
Application.Worksheets("DATA").Range("ProjectName").Value
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Metres"
        .Axes(xlValue, xlPrimary).HasTitle = True
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Deg. Celsius"
    End With
End Sub

Public Sub ModifyPressurePlot()
    Sheets("OP").Select
    With ActiveChart
        .ChartType = xlLine
        .SeriesCollection(1).XValues = "=" & RangePointedToBy("KmPostCell")
        .SeriesCollection(1).Values = "=" & RangeWithTopAt("PRESSURE")
        .SeriesCollection(1).Name = "=""Operating Pressure"""
        .SeriesCollection(2).Values = "=" & RangeWithTopAt("MAOPBARS")
        .SeriesCollection(2).Name = "=""MAOP"""

        .HasTitle = True
        .ChartTitle.Characters.Text = "OPERATING PRESSURE" & "-" &
Application.Worksheets("DATA").Range("ProjectName").Value
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Metres"
        .Axes(xlValue, xlPrimary).HasTitle = True
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Bars"
    End With
```

```
End Sub

Public Sub ModifyStationsPlot()
    Sheets("STATIONS").Select
    With ActiveChart
        .SeriesCollection(1).XValues = "=" & ColumnWithTopAtOfHeight("STATIONS", _
            Application.Worksheets("DATA").Range("NOOFSTATIONS").Cells(1, 1).Value)
        .SeriesCollection(1).Values = "=" & ColumnWithTopAtOfHeight("HEADCHANGE", _
            Application.Worksheets("DATA").Range("NOOFSTATIONS").Cells(1, 1).Value)
        .SeriesCollection(1).Name = "=""Stations"""

        .HasTitle = True
        .ChartTitle.Characters.Text = "PUMP AND REDUCTION STATIONS" & "-" &
Application.Worksheets("DATA").Range("ProjectName").Value
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Metres"
        .Axes(xlValue, xlPrimary).HasTitle = True
        .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Metres"
    End With
End Sub


Sub Macro1()
'
' Macro1 Macro
' Macro recorded 9/2/98 by IT Department
'

'
    ActiveChart.SeriesCollection.NewSeries
    ActiveChart.SeriesCollection(4).Values = "=USER!R2C3:R9C3"
    ActiveChart.SeriesCollection(4).Name = "=""xx"""
End Sub
```

Appendix B. Source Code of the Computations Module

```
Option Base 0

' maximum array index
Public Const MaxIndex = 2000

' Infinity
Public Const Infinity = 1E+20

' gravity acceleration
Public Const gravity = 9.81

' minimum pressure Newton/m2
Public Const MinimumPressure = 500000

' pump maximum discharge pressure Newton/m2
Public Const MaxPumpDisch = 1000000000

' operating pressure / maximum allowed operating pressure
Public Const OpByMaop = 0.909

' pump minimum suction pressure  Newton/m2
Public Const MinPumpSuct = 600372

' the total No of km posts
Public N As Integer
' Pipeline flowrate
Public Q As Double

' oil specific gravity @ various temperatures
Public Ro(0 To 255) As Double
' various temp @ which Ro is given
Public RoTemp(0 To 255) As Double
' no of given Ro values
Public NoOfRoValues As Integer
' oil viscosity various temperatures
Public Nu(0 To 255) As Double
' various temp @ which Nu is given
Public NuTemp(0 To 255) As Double
' no of given Nu values
Public NoOfNuValues As Integer

' Ambient temperature on various segments
Public T(0 To MaxIndex) As Double
' Product temperature on various segments
Public Temp(0 To MaxIndex) As Double
' pipeline design factor used with yield strength
Public DesignFactor(0 To MaxIndex) As Double
' km post (input) m
Public Km(0 To MaxIndex) As Double
' elevation (input) m
```

```
Public El(0 To MaxIndex) As Double
' overall diameter (input) m
Public OD(0 To MaxIndex) As Double
' wall thickness (input) m
Public WT(0 To MaxIndex) As Double
' yield strength (input) psi
Public YS(0 To MaxIndex) As Double
' pipe wall roughness
Public Roughness(0 To MaxIndex) As Double
' Heat Transfer Rate through pipeline surface
Public HeatTransferRate(0 To MaxIndex) As Double
' head (calc) m
Public Head(0 To MaxIndex) As Double
' maximum allowed operating pressure (calc) psi
Public MAOP(0 To MaxIndex) As Double
' maximum allowed operating head (calc) m
Public MAOH(0 To MaxIndex) As Double

' Heat capacity of the product
Public ProductHeatCapacity As Double

' No of pump and reduction stations
Public NoOfStations As Integer
' Pump and reduction stations +1 pump, -1 reduct
Public Station(0 To MaxIndex) As Integer
' Delta head @ each station (m)
Public DeltaH(0 To MaxIndex) As Double
' Station x coordinate (m)
Public StX(0 To MaxIndex) As Double
' upgradable station data
Dim UpgradeNoOfStations As Integer
Dim UpgradeStation(0 To MaxIndex) As Integer
Dim UpgradeStX(0 To MaxIndex) As Double

' matrix M[i,j] contains costs of achieving
' output head in the interval
' [HeadOut(j), HeadOut(j+1)] if the input head
' id in the interval [HeadOut(i), HeadOut(i+1)]
Public M(0 To MaxIndex, 0 To MaxIndex) As Double

' matrix contains input heads for an interval
Public HeadIn(0 To MaxIndex) As Double
' matrix contains output heads for an interval
Public HeadOut(0 To MaxIndex) As Double



Private Declare Sub MessageBeep Lib "User32" (ByVal N As Integer)
Sub CallMyDll()
   Call MessageBeep(0) ' Call Windows DLL procedure.
   MessageBeep 0   ' Call again without Call keyword.
End Sub


' this data can only be called if valid data
```

```
Public Sub LoadData()
Dim i As Integer
'On Error GoTo InvalidData
    ProgressOn
    With Application.Worksheets("DATA")
        N = Val(.Range("NoOfKmPosts").Value)
        If N <= 1 Then
            MsgBox "No of KM posts must be an integer above 1", , "Ordos 99"
            GoTo InvalidData
        End If
        Q = Val(.Range("Q").Value)
        If Q <= 0 Then
            MsgBox "Flow rate must be a positive real", , "Ordos 99"
            GoTo InvalidData
        End If
        ' read table of Ro versus T
        i = 0
        While Not IsEmpty(Range(.Range("SGCell").Value).Cells(1 + i, 1).Value)
            Ro(i) = Val(Range(.Range("SGCell").Value).Cells(1 + i, 1).Value)
            RoTemp(i) = Val(Range(.Range("SGCell").Value).Cells(1 + i, 2).Value)
            If Ro(i) <= 0 Then
                MsgBox "Specific gravity must be a postive real", , "Ordos 99"
                GoTo InvalidData
            End If
            i = i + 1
        Wend
        '  if no table found ...
        If i = 0 Then
            MsgBox "At least one value of specific gravity must be specified", , "Ordos 99"
            GoTo InvalidData
        End If
        NoOfRoValues = i

        ' read table of Nu versus T
        i = 0
        While Not IsEmpty(Range(.Range("KVCell").Value).Cells(1 + i, 1).Value)
            Nu(i) = Val(Range(.Range("KVCell").Value).Cells(1 + i, 1).Value)
            NuTemp(i) = Val(Range(.Range("KVCell").Value).Cells(1 + i, 2).Value)
            If Nu(i) <= 0 Then
                MsgBox "Kinematic viscosity must be a postive real", , "Ordos 99"
                GoTo InvalidData
            End If
            i = i + 1
        Wend
        '  if no table found ...
        If i = 0 Then
            MsgBox "At least one value of kinematic viscosity must be specified", , "Ordos 99"
            GoTo InvalidData
        End If
        NoOfNuValues = i
        TerminalHead = Val(.Range("TerminalHead").Value)
        If TerminalHead <= 0 Then
            MsgBox "Terminal must be a postive real or zero", , "Ordos 99"
           GoTo InvalidData
        End If
```

```
For i = 0 To N - 1
    Temp(i) = T(i) ' initial approximation of product temperature
    Km(i) = Val(Range(.Range("KMPostCell").Value).Cells(1 + i, 1).Value)
    If Km(i) < 0 Then
        MsgBox "KM Posts must be postive reals or zero, index " & _
            Format(i, "####"), , "Ordos 99"
        GoTo InvalidData
    End If
    If i > 0 Then
        If Km(i) <= Km(i - 1) Then
            MsgBox "KM Posts must monotonously increase, index " & _
            Format(i, "####"), , "Ordos 99"
            GoTo InvalidData
        End If
    End If
    El(i) = Val(Range(.Range("ElevationCell").Value).Cells(1 + i, 1).Value)
    OD(i) = Val(Range(.Range("ODCell").Value).Cells(1 + i, 1).Value)
    If OD(i) <= 0 Then
        MsgBox "Overall diameters must be postive reals, index " & _
            Format(i, "####"), , "Ordos 99"
        GoTo InvalidData
    End If
    WT(i) = Val(Range(.Range("WTCell").Value).Cells(1 + i, 1).Value)
    If WT(i) <= 0 Then
        MsgBox "Wall thicknesses must be postive reals, index " & _
            Format(i, "####"), , "Ordos 99"
        GoTo InvalidData
    End If
    YS(i) = Val(Range(.Range("YSCell").Value).Cells(1 + i, 1).Value)
    If YS(i) <= 0 Then
        MsgBox "Yield strengths must be postive reals, index " & _
            Format(i, "####"), , "Ordos 99"
        GoTo InvalidData
    End If

    Roughness(i) = Val(Range(.Range("RoughnessCell").Value).Cells(1 + i, 1).Value)
    If Roughness(i) <= 0 Then
        MsgBox "Roughnesses must be postive reals, index " & _
            Format(i, "####"), , "Ordos 99"
        GoTo InvalidData
    End If

    T(i) = Val(Range(.Range("TemperatureCell").Value).Cells(1 + i, 1).Value)
    DesignFactor(i) = Val(Range(.Range("DesignFactorCell").Value).Cells(1 + i, 1).Value)
    If DesignFactor(i) <= 0 Then
        MsgBox "Design factor be a postive real below or equal 1", , "Ordos 99"
        GoTo InvalidData
    End If

    HeatTransferRate(i) = Val(Range(.Range("HTRCell").Value).Cells(1 + i, 1).Value)
    If HeatTransferRate(i) < 0 Then
        MsgBox "het transfer rates must be postive reals, index " & _
            Format(i, "####"), , "Ordos 99"
        GoTo InvalidData
```

```
            End If
            If i Mod 15 = 0 Then Call ShowProgress("Loading data...", i, N - 1)
        Next i
        ProductHeatCapacity = Val(Range(.Range("HCCell").Value).Cells(1, 1).Value)
        Temp(0) = .Range("InitialTemperature").Value
    End With
    ProgressOff
    Exit Sub
InvalidData: MsgBox "Invalid parameter - check your input data", , "Ordos 99"
Call EraseData
ProgressOff
End Sub


Public Sub EraseData()
    N = 0
End Sub


Public Function DataLoaded() As Boolean
    DataLoaded = (N > 1)
End Function

Public Sub Calc_MaxOpHeadAndPressure()
Dim i As Integer
    Call Calc_ProductTemperature
    If Not DataLoaded Then MsgBox "No data": Exit Sub
    ProgressOn
    For i = 0 To N - 1
        MAOP(i) = 2 * (WT(i) / OD(i)) * YS(i) * gravity * (0.454 / (0.0254 * 0.0254)) *
DesignFactor(i)
        MAOH(i) = MAOP(i) / (Density(i) * gravity) + El(i)
        Application.Worksheets("DATA").Range("MAOP").Cells(1 + i, 1).Value = _
        MAOP(i)
        Application.Worksheets("DATA").Range("MAOPBARS").Cells(1 + i, 1).Value = _
        MAOP(i) / gravity / 10200
        Application.Worksheets("DATA").Range("MAOH").Cells(1 + i, 1).Value = _
        MAOH(i)
        Application.Worksheets("DATA").Range("GRAD").Cells(1 + i, 1).Value = _
        Gradient(i)
        If i Mod 20 = 0 Then _
           Call ShowProgress("Calculating MAOP...", i, N - 1)
    Next i
    Call ModifyPressurePlot
    Call ModifyHeadPlot
    Call ModifyGradientPlot
    Charts("HEAD_AND_EL").Activate
    ProgressOff
End Sub

Public Function CombinedHeatTransfer(ByVal i As Integer) As Double
Dim yield1 As Double
    yield1 = lambda(i) * ProductHeatCapacity * Density(i) * velocity(i) / 8
    CombinedHeatTransfer = 1 / (1 / yield1 + 1 / HeatTransferRate(i))
End Function
```

```
Public Sub Calc_ProductTemperature()
Dim i As Integer
    If Not DataLoaded Then MsgBox "No data": Exit Sub
    ProgressOn
    Application.Worksheets("DATA").Range("TEMP").Cells(1, 1).Value = _
        Temp(0)
    For i = 1 To N
        Temp(i) = Temp(i - 1) + (4 * CombinedHeatTransfer(i - 1) / Diameter(i - 1) _
                * (T(i - 1) - Temp(i - 1)) / velocity(i - 1) + _
                Gradient(i - 1) * Density(i - 1) * gravity) * (Km(i) - Km(i - 1)) / (Density(i - 1) *
ProductHeatCapacity)
        Application.Worksheets("DATA").Range("TEMP").Cells(1 + i, 1).Value = _
            Temp(i)
        If i Mod 20 = 0 Then _
           Call ShowProgress("Calculating Temperature...", i, N - 1)
    Next i
    Call ModifyTemperaturePlot
    Charts("TEMPERATURE").Activate
    ProgressOff
End Sub

Public Sub InstallStations()
Dim i, j As Integer
Dim h As Double
    ProgressOn

    ' load station locations and delta head
    With Application.Worksheets("DATA")
        TerminalHead = Val(.Range("TerminalHead").Value)
        N = Val(.Range("NoOfKmPosts").Value)

        NoOfStations = .Range("NOOFSTATIONS").Cells(1, 1).Value
        For j = 0 To NoOfStations - 1
            StX(j) = .Range("STATIONS").Cells(1 + j, 1).Value
            DeltaH(j) = .Range("DELTAHEAD").Cells(1 + j, 1).Value
            Station(j) = .Range("STATIONTYPES").Cells(1 + j, 1).Value
            If j Mod 5 = 0 Then _
                Call ShowProgress("Loading station data...", j, N - 1)
        Next j
    End With
    ProgressOff
    ProgressOn
    i = N - 1: h = TerminalHead: j = 0
    Do While i >= 0
        Head(i) = h
        Application.Worksheets("DATA").Range("HEAD").Cells(1 + i, 1).Value = _
        h
        Application.Worksheets("DATA").Range("PRESSURE").Cells(1 + i, 1).Value = _
        (h - El(i)) * Density(i) / 10200
        i = i - 1
        If i < 0 Then Exit Do
        h = h + Gradient(i) * (Km(i + 1) - Km(i))
        If j < NoOfStations Then
            Do While Km(i) <= StX(j)
```

```
            h = h - Station(j) * DeltaH(j)
            j = j + 1
            If j = NoOfStations Then Exit Do
        Loop
    End If
    If i Mod 20 = 0 Then _
        Call ShowProgress("Preparing plot data...", N - i, N - 1)
Loop
ProgressOff
Call ModifyHeadPlot
Call ModifyPressurePlot
Call ModifyStationsPlot
Charts("HEAD_AND_EL").Activate
End Sub

Public Sub UpgradeStations()
Dim i As Integer, j As Integer, x As Double, h As Double, Delta As Double
Dim k As Integer, upgrade As Integer
Dim old_h As Double, h1 As Double

    If Not DataLoaded Then MsgBox "No data": Exit Sub
    TerminalHead = Val(Worksheets("DATA").Range("TerminalHead").Value)
    N = Val(Worksheets("DATA").Range("NoOfKmPosts").Value)

    If TerminalHead < Profile(N - 1) Then _
        MsgBox "Destination head must be in excess of elevation", , _
        "Ordos 99": Exit Sub
    ProgressOn
    With Application.Worksheets("DATA")
        UpgradedNoOfStations = .Range("UPGRADENOOFSTATIONS").Cells(1, 1).Value
        For j = 0 To UpgradedNoOfStations - 1
            UpgradeStX(j) = .Range("UPGRADESTATIONSX").Cells(1 + j, 1).Value
            UpgradeStation(j) = .Range("UPGRADESTATIONTYPES").Cells(1 + j, 1).Value
            Call ShowProgress("Loading upgraded station data...", j + 1, UpgradedNoOfStations)
        Next j
    End With

    ProgressOff
    ProgressOn
    NoOfStations = 0: h = TerminalHead: i = N - 1: x = Km(N - 1)
    upgrade = 0
    ' x = latest studied node co-ordinate
    While i > 0
        i = i - 1
        h_old = h
        h = h + Gradient(i) * (x - Km(i))
        If Km(i) <= UpgradeStX(upgrade) And upgrade < UpgradedNoOfStations Then
            x = UpgradeStX(upgrade)
            h = h - Gradient(i) * (x - Km(i))
            upgrade = upgrade + 1
            If UpgradeStation(upgrade - 1) = 1 Then ' upgraded pump
                Delta = h
                GoTo AddPump
            Else ' upgraded regulator
                GoTo AddRegulator
```

```
        End If
      ElseIf h <= Profile(i) Then ' run into ground - regulator site
        x = IntersectionOf(Km(i), Km(i + 1), Profile(i), Profile(i + 1), Km(i), x, h, h_old)

        ' equate h to just elevation of point x on the profile
        h = Profile(i) + (x - Km(i)) * (Profile(i + 1) - Profile(i)) / (Km(i + 1) - Km(i))
        ' add a reduction station
AddRegulator:
        StX(NoOfStations) = x: Station(NoOfStations) = -1

        j = HighPoint(x, h)

        If j = -1 Then MsgBox "Error High Point", , "Ordos 99": Exit Sub

        Delta = 0
        If j < i Then
          For k = j To i - 1
            Delta = Delta + (Km(k + 1) - Km(k)) * Gradient(k)
          Next k
        End If
        Delta = Delta + (x - Km(i)) * Gradient(i)

        ' profile too high
        'If Profile(El(j)) > Operating(j) Then
        '    MsgBox "Elevation at point " & Format(Km(j), "##.##") & " exceeds OP", , _
        '    "Ordos 99 - calculation aborted"
        '    ProgressOff
        '    Exit Sub
        'End If

        ' head reduction
        DeltaH(NoOfStations) = Profile(j) - Delta - h
        h = h + DeltaH(NoOfStations)
        NoOfStations = NoOfStations + 1
        h = h + 0.01 ' margin
      ElseIf h >= PumpOrOp(i) Then ' exceeded OP - pump site
        x = IntersectionOf(Km(i), Km(i + 1), _
        PumpOrOp(i), PumpOrOp(i + 1), _
             Km(i), x, h, h_old)

        Delta = PumpOrOp(i) + (x - Km(i)) * _
            (PumpOrOp(i + 1) - PumpOrOp(i)) / _
            (Km(i + 1) - Km(i))
AddPump:
        ' minimum suction pressure
        h = Profile(i) + (x - Km(i)) * _
            (Profile(i + 1) - Profile(i)) / _
            (Km(i + 1) - Km(i))
        h = Maximum(h, El(i) + MinPumpSuct / (Density(i) * gravity))
        StX(NoOfStations) = x

        j = HighPoint(x, h)

        If j <> -1 Then
          h1 = Profile(j)
```

```
            If j < i Then
                For k = j To i - 1
                    h1 = h1 - (Km(k + 1) - Km(k)) * Gradient(k)
                Next k
            End If
            h1 = h1 - (x - Km(i)) * Gradient(i)
            If h1 > h Then h = h1
        End If

        Delta = Delta - h

        ' add a pump station
        Station(NoOfStations) = 1
        DeltaH(NoOfStations) = Delta
        NoOfStations = NoOfStations + 1
        h = h + 0.01 ' margin
    Else ' carry on OK
        x = Km(i)
    End If
    If i Mod 2 = 0 Then _
        Call ShowProgress("Upgrading station layout...", N - i, N - 1)
Wend
ProgressOff
ProgressOn
' save station locations and delta head
With Application.Worksheets("DATA")
    .Range("NOOFSTATIONS").Cells(1, 1).Value = _
            NoOfStations
    For j = 0 To NoOfStations - 1
        .Range("STATIONS").Cells(1 + j, 1).Value = _
            StX(j)
        .Range("DELTAHEAD").Cells(1 + j, 1).Value = _
            DeltaH(j)
        .Range("STATIONTYPES").Cells(1 + j, 1).Value = _
            Station(j)
    If j Mod 5 = 0 Then _
        Call ShowProgress("Storing station data...", j, N - 1)
    Next j
End With

ProgressOff
Call InstallStations
End Sub

Public Sub LocateStations()
Dim i As Integer, j As Integer, x As Double, h As Double, Delta As Double
Dim k As Integer
Dim old_h As Double

    If Not DataLoaded Then MsgBox "No data": Exit Sub
    Calc_MaxOpHeadAndPressure
    TerminalHead = Val(Worksheets("DATA").Range("TerminalHead").Value)
    N = Val(Worksheets("DATA").Range("NoOfKmPosts").Value)

    If TerminalHead < Profile(N - 1) Then _
```

```
    MsgBox "Destination head must be in excess of elevation", , _
    "Ordos 99": Exit Sub

ProgressOn
NoOfStations = 0: h = TerminalHead: i = N - 1: x = Km(N - 1)
' x = latest installed station co-ordinate
While i > 0
    i = i - 1
    h_old = h
    h = h + Gradient(i) * (x - Km(i))
    If h <= Profile(i) Then ' run into ground - regulator site
        x = IntersectionOf(Km(i), Km(i + 1), Profile(i), Profile(i + 1), Km(i), x, h, h_old)
        ' equate h to just elevation of point x on the profile
        h = Profile(i) + (x - Km(i)) * (Profile(i + 1) - Profile(i)) / (Km(i + 1) - Km(i))
        ' add a reduction station
        StX(NoOfStations) = x: Station(NoOfStations) = -1

        j = HighPoint(x, h)

        If j = -1 Then MsgBox "Error High Point", , "Ordos 99": Exit Sub

        Delta = 0
        If j < i Then
            For k = j To i - 1
                Delta = Delta + (Km(k + 1) - Km(k)) * Gradient(k)
            Next k
        End If
        Delta = Delta + (x - Km(i)) * Gradient(i)

        ' profile too high
        'If Profile(El(j)) > Operating(j) Then
        '    MsgBox "Elevation at point " & Format(Km(j), "##.##") & " exceeds OP", , _
        '    "Ordos 99 - calculation aborted"
        '    ProgressOff
        '    Exit Sub
        'End If

        ' head reduction
        DeltaH(NoOfStations) = Profile(j) - Delta - h
        NoOfStations = NoOfStations + 1
        i = j
        h = Profile(j) + 0.01
        x = Km(j)
    ElseIf h >= PumpOrOp(i) Then ' exceeded OP - pump site
        x = IntersectionOf(Km(i), Km(i + 1), _
        PumpOrOp(i), PumpOrOp(i + 1), _
                Km(i), x, h, h_old)

        ' minimum suction pressure
        h = Profile(i) + (x - Km(i)) * _
              (Profile(i + 1) - Profile(i)) / _
              (Km(i + 1) - Km(i))
        h = Maximum(h, El(i) + MinPumpSuct / (Density(i) * gravity))
        Delta = PumpOrOp(i) + (x - Km(i)) * _
              (PumpOrOp(i + 1) - PumpOrOp(i)) / _
```

```
            (Km(i + 1) - Km(i)) - h
        StX(NoOfStations) = x

        j = HighPoint(x, h)

        If j <> -1 Then
            h = Profile(j)
            If j < i Then
                For k = j To i - 1
                    h = h - (Km(k + 1) - Km(k)) * Gradient(k)
                Next k
            End If
            h = h - (x - Km(i)) * Gradient(i)
            Delta = PumpOrOp(i) + (x - Km(i)) * _
               (PumpOrOp(i + 1) - PumpOrOp(i)) / _
               (Km(i + 1) - Km(i)) - h
        End If
        ' add a pump station
        Station(NoOfStations) = 1
        DeltaH(NoOfStations) = Delta
        NoOfStations = NoOfStations + 1
        h = h + 0.01 'extra margin
    Else ' carry on OK
        x = Km(i)
    End If
    If i Mod 15 = 0 Then _
        Call ShowProgress("Locating stations...", N - i, N - 1)
Wend
ProgressOff
ProgressOn
' save station locations and delta head
With Application.Worksheets("DATA")
    .Range("NOOFSTATIONS").Cells(1, 1).Value = _
            NoOfStations
    For j = 0 To NoOfStations - 1
        .Range("STATIONS").Cells(1 + j, 1).Value = _
            StX(j)
        .Range("DELTAHEAD").Cells(1 + j, 1).Value = _
            DeltaH(j)
        .Range("STATIONTYPES").Cells(1 + j, 1).Value = _
            Station(j)
    If j Mod 5 = 0 Then _
        Call ShowProgress("Storing station data...", j, N - 1)
    Next j
End With

ProgressOff
Call InstallStations
End Sub


Public Function Profile(ByVal i As Integer) As Double
    Profile = El(i) + MinimumPressure / (Density(i) * gravity)
End Function
```

```
Public Function Operating(ByVal i As Integer) As Double
   Operating = El(i) + OpByMaop * MAOP(i) / (Density(i) * gravity)
End Function

Public Function Diameter(ByVal i As Integer) As Double
   Diameter = OD(i) - 2 * WT(i)
End Function

Public Function velocity(ByVal i As Integer) As Double
   velocity = 4 * Q / (3.1415 * Diameter(i) * Diameter(i))
End Function

Public Function viscosity(ByVal i As Integer) As Double
Dim j As Integer
   If NoOfNuValues = 1 Then
      viscosity = Nu(0)
   Else
      j = 0
      Do While Temp(i) > NuTemp(j)
         j = j + 1
         If j = NoOfNuValues Then Exit Do
      Loop
      If j = NoOfNuValues Then
         viscosity = Nu(j - 1) + (Nu(j - 1) - Nu(j - 2)) * (Temp(i) - NuTemp(j - 1)) / (NuTemp(j - 1)
- NuTemp(j - 2))
      ElseIf j = 0 Then
         viscosity = Nu(j) + (Nu(j + 1) - Nu(j)) * (Temp(i) - NuTemp(j)) / (NuTemp(j + 1) -
NuTemp(j))
      Else
         j = j - 1
         viscosity = Nu(j) + (Nu(j + 1) - Nu(j)) * (Temp(i) - NuTemp(j)) / (NuTemp(j + 1) -
NuTemp(j))
      End If
   End If
End Function


Public Function Density(ByVal i As Integer) As Double
Dim j As Integer
   If NoOfRoValues = 1 Then
      Density = Ro(0)
   Else
      j = 0
      Do While Temp(i) > RoTemp(j)
         j = j + 1
         If j = NoOfRoValues Then Exit Do
      Loop
      If j = NoOfRoValues Then
         Density = Ro(j - 1) + (Ro(j - 1) - Ro(j - 2)) * (Temp(i) - RoTemp(j - 1)) / (RoTemp(j - 1) -
RoTemp(j - 2))
      ElseIf j = 0 Then
         Density = Ro(j) + (Ro(j + 1) - Ro(j)) * (Temp(i) - RoTemp(j)) / (RoTemp(j + 1) -
RoTemp(j))
      Else
         j = j - 1
```

```
            Density = Ro(j) + (Ro(j + 1) - Ro(j)) * (Temp(i) - RoTemp(j)) / (RoTemp(j + 1) -
RoTemp(j))
        End If
    End If
End Function


Public Function Re(ByVal i As Integer) As Double
    Re = velocity(i) * Diameter(i) / viscosity(i)
End Function

Public Function Sqrt(ByVal x As Double) As Double
    Sqrt = Exp(0.5 * Application.Ln(Abs(x)))
End Function

Public Function F_lambda(ByVal s As Double, ByVal i As Integer) As Double
F_lambda = Application.Ln(Roughness(i) / (3.7 * Diameter(i)) + 2.51 * Abs(s) / Re(i)) + _
           Application.Ln(10) * s / 2
End Function

Public Function dF_lambda_ds(ByVal s As Double, ByVal i As Integer) As Double
    dF_lambda_ds = 2.51 / (Roughness(i) * Re(i) / (3.7 * Diameter(i)) + 2.51 * Abs(s)) + _
                   Application.Ln(10) / 2
End Function


Public Function lambda(ByVal i As Integer) As Double
Dim s As Double, j As Integer
'    lambda = 0.3164 / Exp(0.25 * Application.Ln(Re(i)))
'    lambda = 0.0096 + 5.7 * Sqrt(Roughness(i) _
'    / Diameter(i)) + 1.7 * Sqrt(1 / Re(i))
    lambda = 0.11 * Exp(0.2 * Application.Ln(58 / Re(i) + 2 * Roughness(i) / Diameter(i)))
    s = 1 / Sqrt(lambda)
    For j = 1 To 5
        s = s - F_lambda(s, i) / dF_lambda_ds(s, i)
    Next j
    lambda = 1 / (s * s)
End Function

Public Function Gradient(ByVal i As Integer) As Double
    Gradient = lambda(i) * (velocity(i) * velocity(i)) / (2 * Diameter(i) * gravity)
End Function

Public Function IntersectionOf(ByVal x11 As Double, ByVal x12 As Double, _
        ByVal y11 As Double, ByVal y12 As Double, _
        ByVal x21 As Double, ByVal x22 As Double, _
        ByVal y21 As Double, ByVal y22 As Double) As Double

Dim k1 As Double, k2 As Double

    If x11 = x12 Or x21 = x22 Then

        If x21 <> x22 Then
            IntersectionOf = x11
        ElseIf x11 <> x12 Then
```

```
        IntersectionOf = x21
    ElseIf x12 <> x21 Then
        IntersectionOf = Infinity
    Else
            Intersectof = x11
    End If
    Exit Function
End If

k1 = (y12 - y11) / (x12 - x11)
k2 = (y22 - y21) / (x22 - x21)

If k1 = k2 Then
    IntersectionOf = Infinity
Else
    IntersectionOf = (x11 * k1 - x21 * k2 + y21 - y11) / (k1 - k2)
End If

End Function


Public Function HighPoint(ByVal x As Double, ByVal y As Double) As Integer
Dim i As Integer, j As Integer, w As Double, hp As Integer, max As Double
    max = 0: i = 0
    Do While Km(i) < x
        i = i + 1
        If i = N Then HighPoint = -1: Exit Function
    Loop
    i = i - 1
    If i = -1 Then HighPoint = -1: Exit Function

    For j = 0 To i
        w = Profile(j)
        For k = j To i - 1
            If w > Operating(k) Then
                w = -Infinity: Exit For
            ElseIf w < Profile(k) Then
                w = -Infinity: Exit For
            Else
                w = w - Gradient(k) * (Km(k + 1) - Km(k))
            End If
        Next k
        w = w - Gradient(i) * (x - Km(i))
        If w > max Then
            max = w
            hp = j
        End If
    Next j

    If max < y Then HighPoint = -1 Else HighPoint = hp

End Function

Public Function Minimum(ByVal a As Double, ByVal b As Double) As Double
    If a < b Then Minimum = a Else Minimum = b
```

```
End Function

Public Function Maximum(ByVal a As Double, ByVal b As Double) As Double
    If a > b Then Maximum = a Else Maximum = b
End Function


Public Function PumpOrOp(ByVal i As Integer) As Double
    PumpOrOp = Minimum(Operating(i), El(i) + MaxPumpDisch / (Density(i) * gravity))
End Function
```

References

[M] M. A. Maharramov. Steady-State and Transient Flows in Hydrocarbon Pipelines. London, O.R.E.M. 2002, Eng. Memo No 29789.

[S] L. I. Sedov. Mechanics of Continua. Moscow, NAUKA, 1973, v I, II.

[DH] J. W. Daily, D.R.F. Harleman. Fluid Dynamics. Addison-Wesley, 1966.