Imperial College of Science, Technology and Medicine

1999/2000 Concrete Structures MSc/DIC Course

Short And Long Term Deformation And Stressing Of Slender Cylindrical Concrete Piers Due To Solar Heating

Dissertation

By Aristotelis E. Charalampakis

Supervisor: Prof. G. L. England

June 2000

Table Of Contents

1. Introduction

1.1 Synopsis

The effect of non – uniform solar heating on slender cylindrical concrete piers is contemplated. Short term, i.e. thermoelastic solutions are obtained, based on a suitable discretization of the structure and a flexible, yet robust description of temperature variations. Long term, i.e. creep solutions are also obtained, based on numerical step – by – step creep analysis and Maxwell creep law.

In order to obtain numerical results, a computer program was developed as part of this dissertation. The program provides user – friendly interface and visualisation of the results. The advantages of numerical step – by – step creep analysis are demonstrated.

Finally, a numerical example demonstrates the significance of the effect of solar heating on slender cylindrical concrete piers, especially for areas where there is a lot of sunshine.

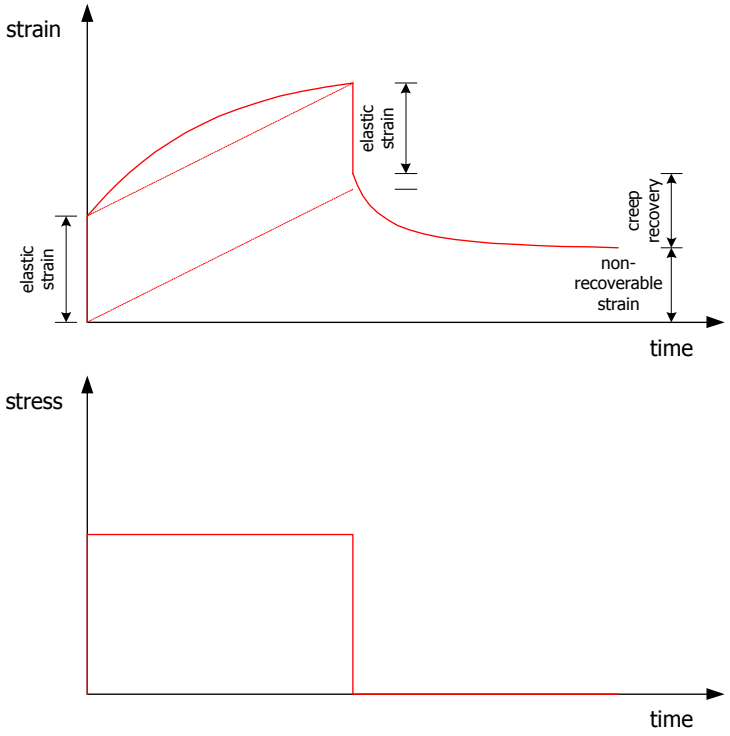
1.2 Acknowledgements

The author would like to thank professor G. L. England for his help in the making of this dissertation.

2. Creep of Concrete

2.1 Definition

Creep of concrete, resulting from the action of a *sustained* stress, is a gradual increase in strain with time. As defined, creep does not include any immediate elastic strains caused by loading or temperature changes, or any shrinkage or swelling caused by moisture changes.



If the sustained load is removed, the strain decreases immediately by an amount equal to the elastic strain at the given age; this is generally lower than the elastic strain on loading since the elastic modulus has increased in the intervening period. This instantaneous recovery is followed by a gradual decrease in strain, called *creep recovery*. This recovery is not complete because creep is not a fully reversible phenomenon.

Creep can be several times as large as the elastic strain on loading. Adding normal drying shrinkage to this and it is clear that these factors can cause considerable deformation and that they are of great importance in structural mechanics.

2.2 Types of creep

Experimental evidence shows that the creep of drying concrete is much greater than the creep of the same concrete in the wet state. Therefore, we can distinguish two types of creep:

- **Basic creep** is defined as the creep that occurs under conditions in which there is no moisture movement between concrete and the environment.
- **Drying creep** is the additional creep that occurs when the concrete is allowed to dry during the period under the load.

2.3 Causes of creep

Creep is associated with physical and chemical processes on the molecular scale. However, there is no convincing *direct* evidence of the actual mechanism of creep.

Tests have shown that concrete from which all evaporable water has been removed exhibits practically no creep. Moreover, creep can occur in mass concrete, therefore seepage of water to the environment is not essential to the basic creep. Hence, internal seepage of water from adsorbed layers to voids such as capillary voids might be the primary mechanism of creep. Indirect evidence is given by the relation between creep and the strength of hydrated cement paste.

Creep after as many as thirty years has been recorded. Therefore, it is probable that the slow, long – term part of creep is due to causes other than the seepage. However, the deformation can only occur in the presence of some evaporable water and this suggests viscous flow or sliding between the gel particles. Such mechanisms are compatible with the influence of temperature on creep, and can explain the largely irreversible character of long – term creep.

2.4 Factors influencing creep

The main factors influencing creep are:

- Temperature.
- Humidity.
- Cement content.
- Water to cement (W/C) ratio.
- Aggregate stiffness and absorption.
- Volume of aggregate.
- Intensity of applied stress.
- Age of concrete.

2.5 Effects of creep

As far as the structural performance is concerned, some effects of creep are beneficial while others are adverse.

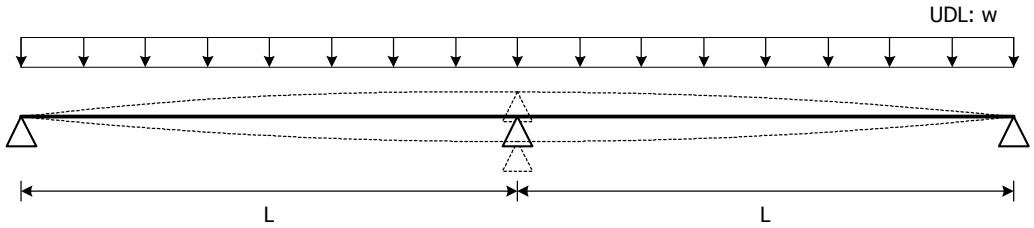
The loss of prestress due to creep is well known and accounted for the failure of all early attempts at prestressing. Only with the introduction of high tensile steel did prestressing become a successful operation.

Creep of concrete does not by itself affect strength, although under very high stresses creep hastens the approach of the limiting strain at which failure takes place.

Moreover, in most cases, the deformations increase considerably with time and this may be a critical consideration in design. Also, in eccentrically loaded columns, creep increases the deflections and can lead to buckling.

On the other hand, creep, unlike shrinkage, is beneficial in relieving stress concentrations induced by shrinkage, temperature changes or movement of supports and has contributed to the success of concrete as a structural material.

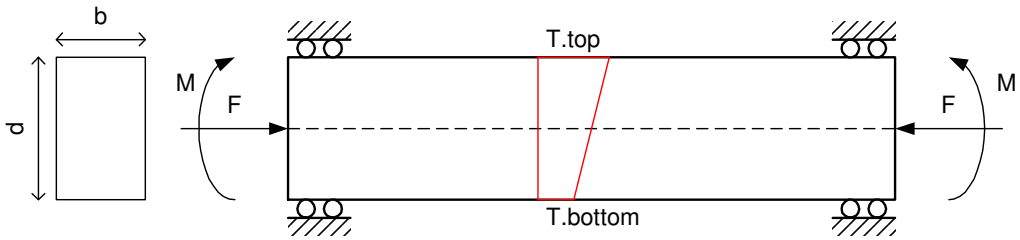
A typical example is a uniformly loaded two – span beam in which the central support is given a vertical sustained displacement. We assume that the temperature is uniform:



It can be shown that, no matter the magnitude of the displacement or if the displacement was upwards or downwards, the value of the bending moment at the central support will converge, with time, to that of the structure with co – linear supports.

Something not generally known is that this effect of creep might be adverse in case of non – uniform distribution of temperatures in slender prestressed structures. A typical example is a flexurally restrained prestressed

concrete beam of rectangular cross-section which carries an axial prestressing force at the centroid of the section and which is also subjected to a sustained temperature crossfall, as shown in the figure:



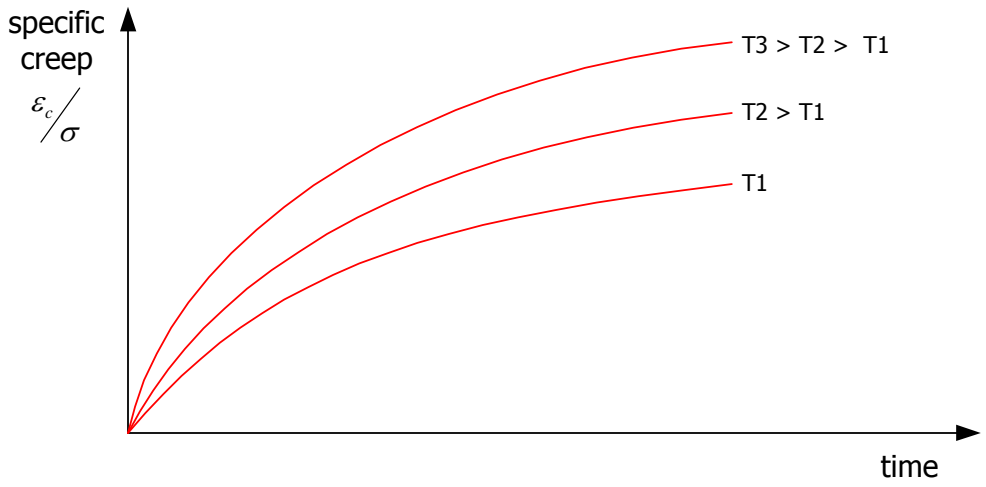
It can be shown that, under some conditions, creep will lead not only to a change of magnitude but also to a change *in sense* of the foundation bending moments.

3. Creep Laws

3.1 Normalisation of creep data

In order to proceed to creep analyses we need to normalise creep strains with respect to these factors that might vary within the structure, such as stress and temperature. In this way, we will be able to use *one simple normalised creep curve* that corresponds to a specific concrete.

We observe that creep strains are *directly* proportional to concrete stress. Therefore, we can first define *specific creep*, i.e. creep strain per unit stress:

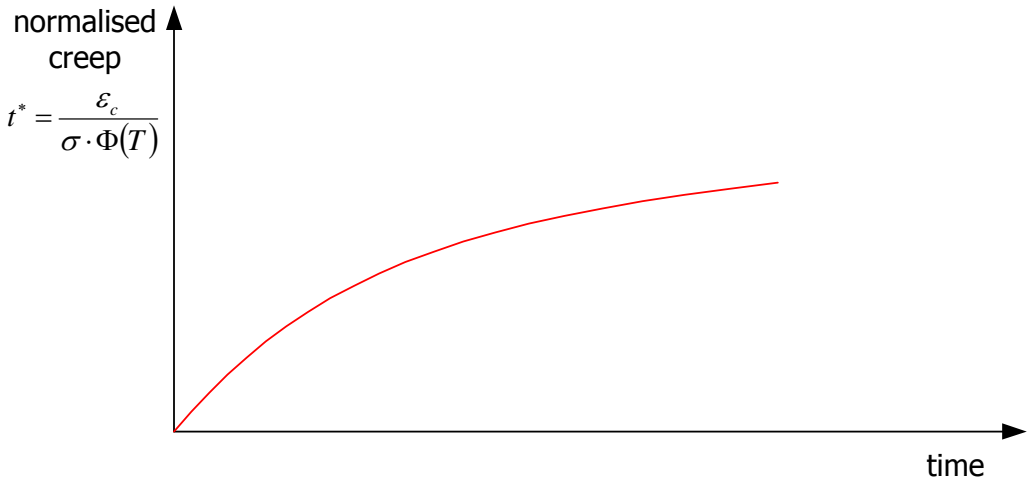


Also, for the same concrete under the same sustained stress, creep increases as temperature increases. In general, creep strains are not directly proportional to temperature. Therefore, we can further normalise the above diagram using *a normalising temperature function, Φ(T)*. This function is usually of the following form:

$$\Phi(T) = a \cdot T^2 + b \cdot T + c$$

Where a, b, c are constants obtained from curve fitting of experimental data, T in degrees Celcius, Φ(T) in degrees Celcius. A simple function such as Φ(T)=T or Φ(T)=T+c is often used.

We can now define *normalised creep*:



Normalised creep (t^*) against real time is a single curve for each concrete which can be used, in theory, for all temperatures and stresses. In reality, this normalisation is valid for stresses less than 40% of ultimate and temperatures in the range of 10 to 140 ^{0}C.

3.2 Viscoelastic models

3.2.1 General considerations

Several viscoelastic models have been used in order to investigate the behaviour of creep. These models are combinations of two main elements:

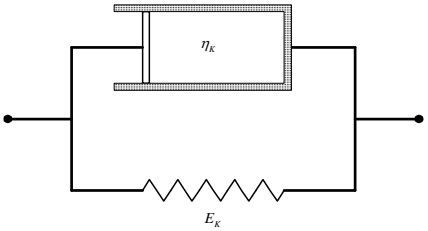
- A **dashpot** with viscosity η:

- A **spring** with elastic modulus E:

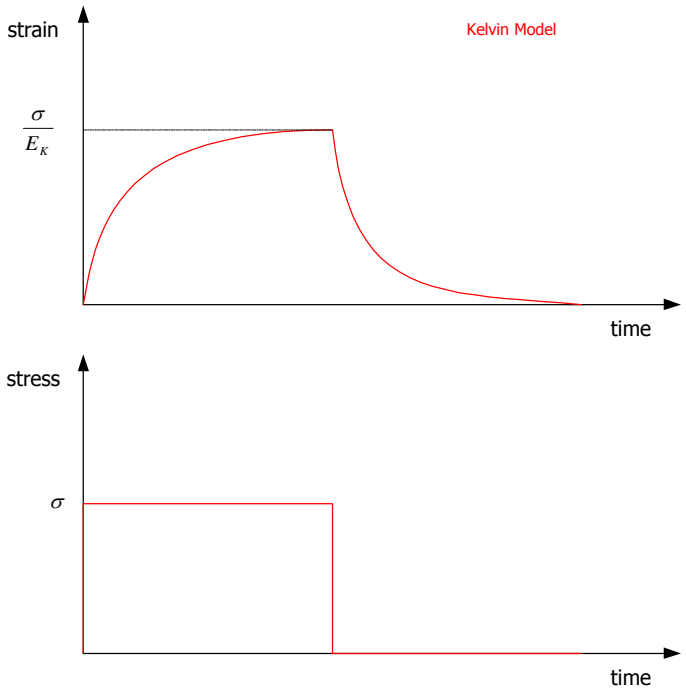
Some creep laws such as *Rate of Creep* or *Rate of Flow* are based on such viscoelastic models.

3.2.2 Kelvin model

It consists of a spring and a dashpot connected in parallel:

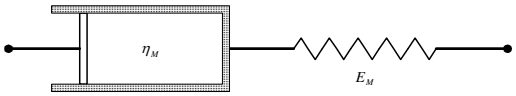


The strain against time diagram for this model is shown in the next figure:

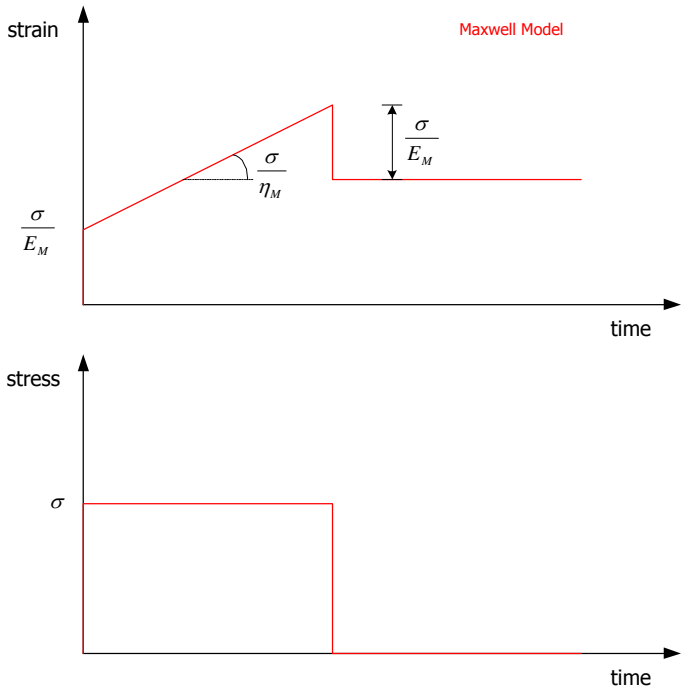


3.2.3 Maxwell model

It consists of a spring and a dashpot connected in series:

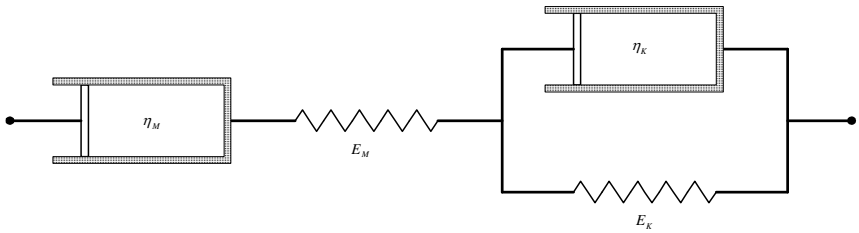


The strain against time diagram for this model is shown in the next figure:

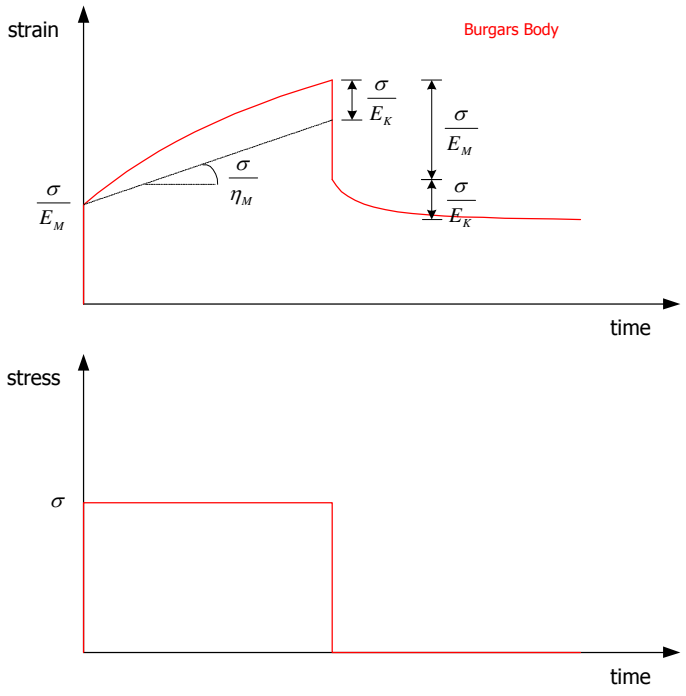


3.2.4 Burgars body

It consists of a Kelvin and a Maxwell model connected in series:



The strain against time diagram for this model is shown in the next figure:



3.3 Creep laws

3.3.1 Effective Modulus

Effective Modulus uses total creep strain data. Effective modulus (E^*) is defined as stress over *total* strain:

$$E^* = \frac{\sigma}{\varepsilon}$$

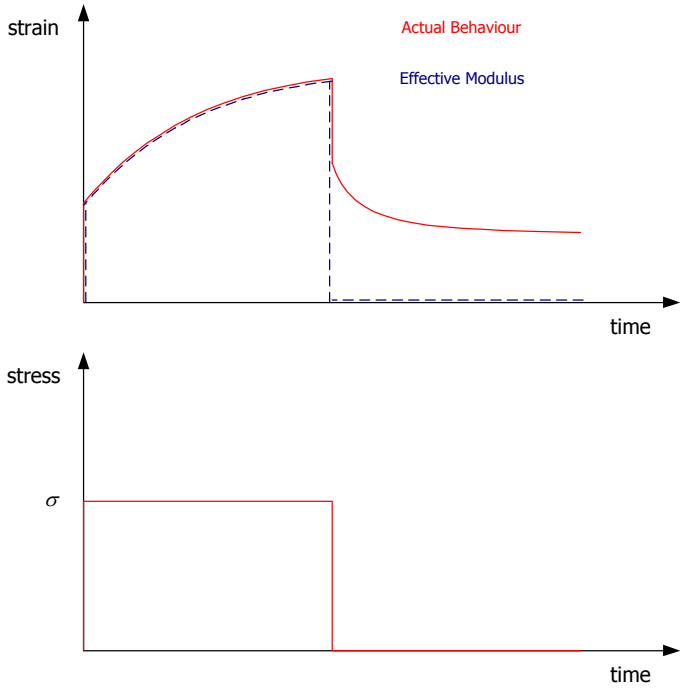
This equation becomes:

$$E^* = \frac{\sigma}{\varepsilon} = \frac{\sigma}{\varepsilon_e + \varepsilon_c} = \frac{\sigma / \varepsilon_e}{1 + \varepsilon_c / \varepsilon_e} = \frac{E}{1+\psi}$$

Where ψ is the *creep coefficient.* We can further relate this coefficient to the normalised creep, as defined earlier:

$$\psi = \frac{\varepsilon_c}{\varepsilon_e} = \frac{\sigma \cdot \Phi(T) \cdot t^*}{\varepsilon_e} = E \cdot \Phi(T) \cdot t^*$$

This creep law is widely used in codes because it is very easy to use; we can obtain the creep solution at a specific time from the normal elastic solution simply by substituting the elastic modulus with the effective modulus at that time. The main disadvantage of this creep law is that there is full strain recovery upon unloading:



3.3.2 Rate of Creep

Rate of Creep uses total creep strain data. We assume that the creep rate is proportional to stress and normalised temperature:

$$t^* = \frac{\varepsilon_c}{\sigma \cdot \Phi(T)} \Rightarrow$$

$$\frac{d\varepsilon_c}{dt} = \sigma \cdot \Phi(T) \cdot \frac{dt^*}{dt}$$

Also, the elastic strain is:

$$\varepsilon_e = \frac{\sigma}{E} \Rightarrow$$

$$\frac{d\varepsilon_e}{dt} = \frac{1}{E} \cdot \frac{d\sigma}{dt}$$

Therefore, the total strain rate is:

$$\varepsilon = \varepsilon_e + \varepsilon_c \Rightarrow$$

$$\frac{d\varepsilon}{dt} = \frac{d\varepsilon_e}{dt} + \frac{d\varepsilon_c}{dt} \Rightarrow$$

$$\frac{d\varepsilon}{dt} = \frac{1}{E} \cdot \frac{d\sigma}{dt} + \sigma \cdot \Phi(T) \cdot \frac{dt^*}{dt} \Rightarrow$$

$$\frac{d\varepsilon}{dt^*} = \frac{1}{E} \cdot \frac{d\sigma}{dt^*} + \sigma \cdot \Phi(T)$$

Or:

$$\dot{\varepsilon} = \frac{\dot{\sigma}}{E} + \sigma \cdot \Phi(T)$$

Where $(\dot{\cdot}) = \frac{d}{dt^*}$

This Rate of Creep law is of identical form to the governing viscoelastic equation for the Maxwell model in real time:

$$\varepsilon = \frac{\sigma}{E_M} + \int \left(\frac{\sigma}{\eta_M} \right) dt \Rightarrow$$

$$\frac{d\varepsilon}{dt} = \frac{1}{E_M} \cdot \frac{d\sigma}{dt} + \frac{\sigma}{\eta_M}$$

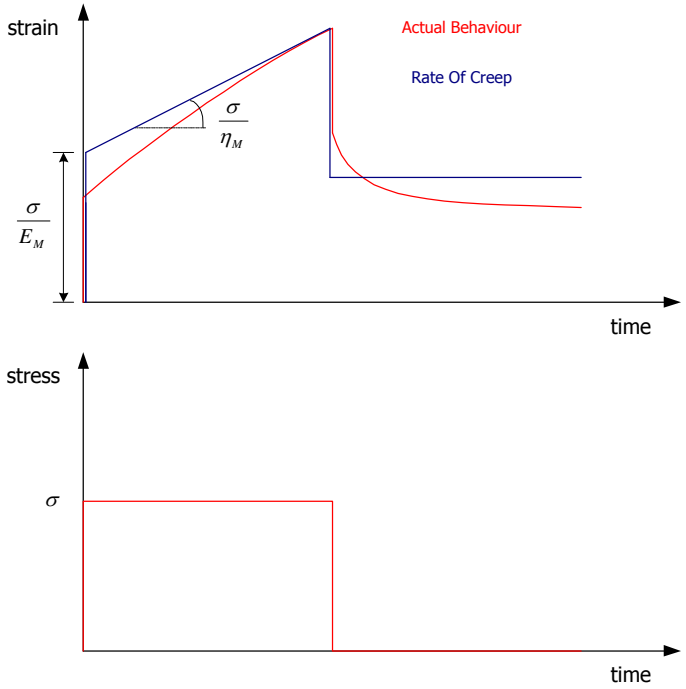
The two equations are equivalent if we make the following substitutions:

$$t \leftrightarrow t^*$$

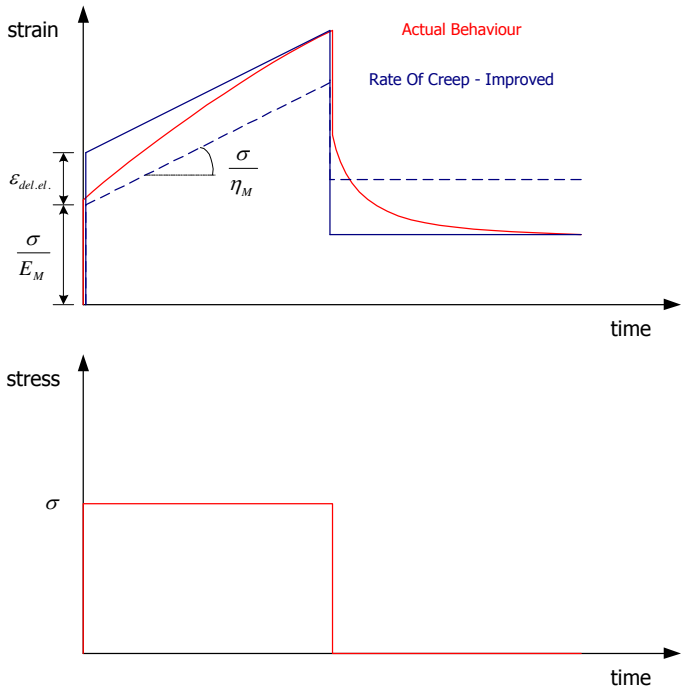
$$\eta \leftrightarrow \frac{1}{\Phi(T)}$$

Normalised creep, t^*, is now defined as *pseudo – time.*

Rate of Creep is easy to formulate and can be easily used with both numerical step – by – step and semi – analytical creep analyses, as described later. The main disadvantage of this law is that there is no creep recovery:

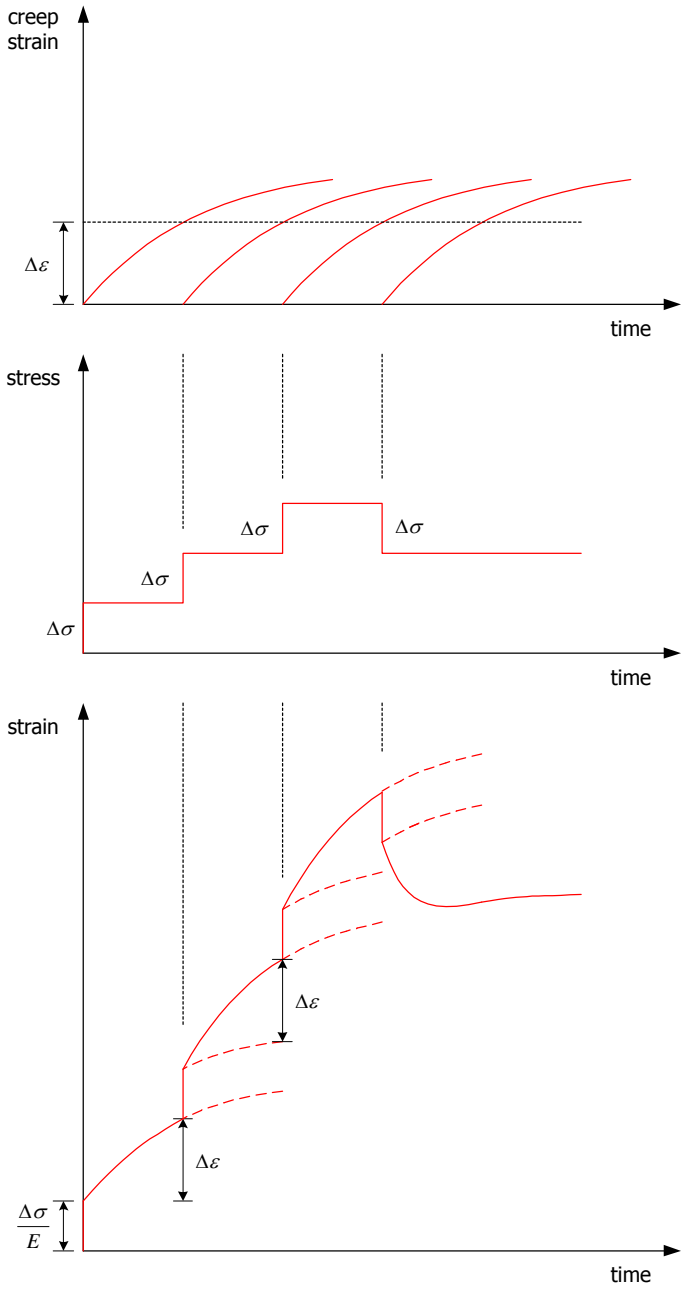


However, we can improve the performance by adding the delayed elastic strain to the initial elastic strain. In this way we obtain better long – term solutions:



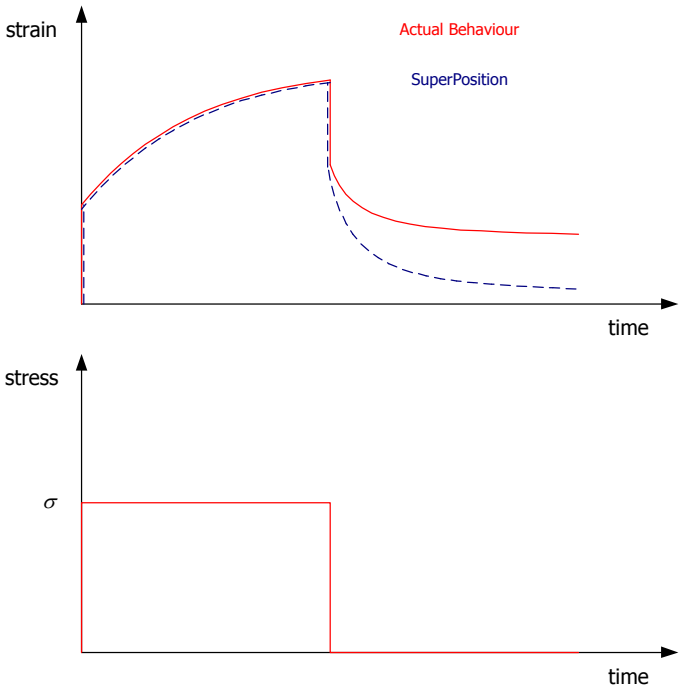
3.3.3 Superposition

As the title implies, this creep law is based on the superposition of diagrams. These diagrams represent the development of creep strain with time, for a specific stress change at a specific time. Therefore, for the same change in stress (Δσ) and if we don't take into account the *ageing* of concrete, all curves will be identical in shape and shifted to the right:



In this case, if we unload everything and allow $t\to\infty$ then $\varepsilon\to 0$.

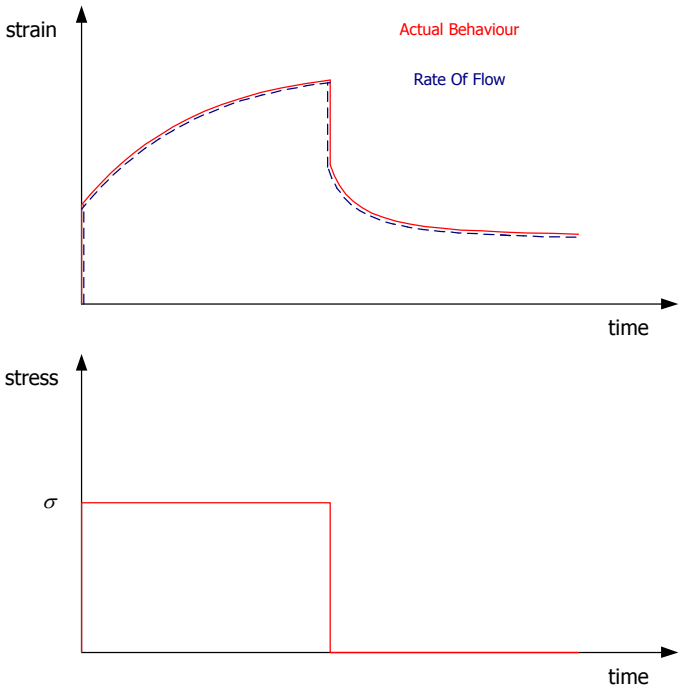
If we take into account the ageing of concrete then the diagrams will not be identical, but the rest of the analysis is the same. However, if $t\to\infty$, then, in general, ε will not converge to zero because the diagrams themselves include some elastic recovery. As a result, this creep law *overestimates* the creep recovery:



Another disadvantage is that it the analysis becomes easily very complicated, while this creep law has nothing to offer more in comparison with the Rate of Flow creep law.

3.3.4 Rate of Flow

Rate of Flow uses flow strain and delayed elastic strain. This creep law is based on Burgars body and can be used with both numerical step – by – step and semi – analytical creep analyses. Although not so easy to formulate, this creep law provides the best results:


strain
Actual Behaviour
Rate Of Flow
time
stress
σ
time

4. Creep Analyses

4.1 Numerical step – by – step

4.1.1 General considerations

As the title implies, this analysis is based on the numerical step – by – step creep analysis of a structure. This kind of analysis is ideally suited for use on computers as it entails much computational effort.

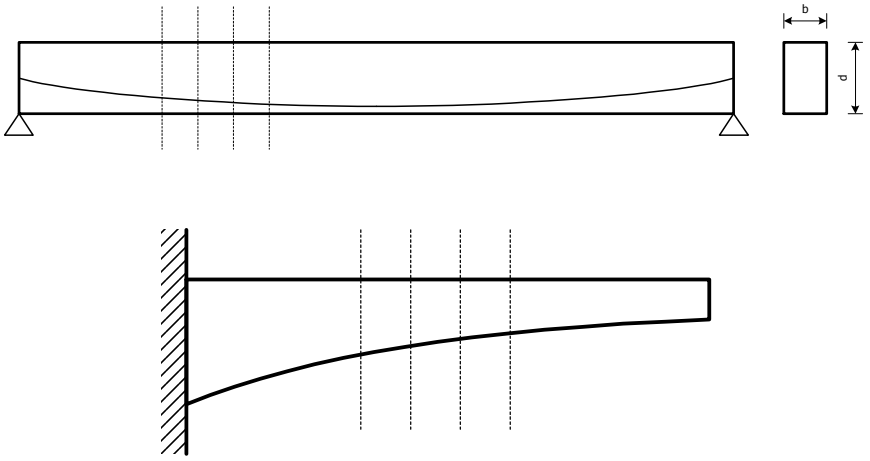
This analysis can accommodate the following creep laws:

- Effective Modulus
- Rate of Creep
- Rate of Flow

As the main part of this dissertation is based on the numerical step – by – step creep analysis using Rate of Creep, this combination will be described in more detail.

4.1.2 Rate Of Creep

We will present this analysis for rectangular beam – type elements of statically determinant structures. These elements are subdivided longitudinally into smaller units and further subdivided into a number of slices in depth:



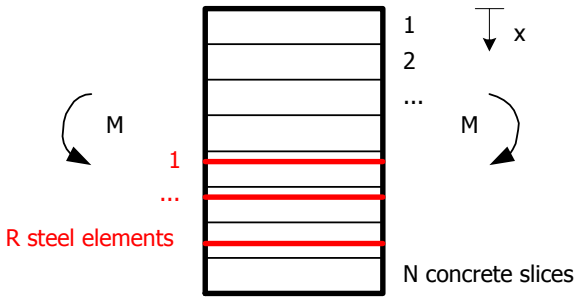
Equilibrium equations are then established between the unit actions, i.e. bending moment and axial force, and the unit stresses in depth. Compatibility is achieved by assuming that *plane sections remain plane at all times*.

Thermal, creep, shrinkage and steel relaxation strains are included at each step of the analysis in time and the analysis is repeated as an 'initial' strain problem with updated initial strains at each step.

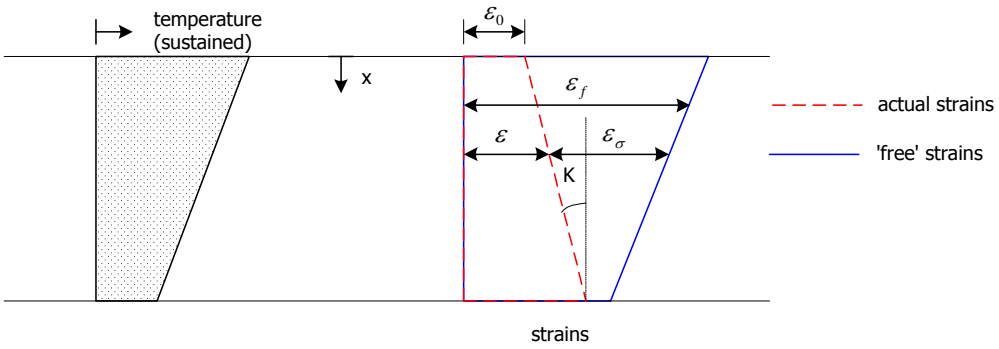
Creep strains are computed incrementally on the assumption that they develop during a short time step *at constant stress*. This stress is the solution from the previous time step.

We can distinguish two cases:

- **Analysis (a):** Only the bending moment and the axial force at t=0 is specified for each unit. The axial force at all other times is an output. This implies that the axial force is not a *follow – up* load, i.e. it changes with time.



Analysis (a)



We assume that the temperature crossfall is sustained. The variation of temperature leads to thermal strains:

$$\Delta\varepsilon_a = \overline{a} \cdot \Delta T \quad (4.1)$$

Where:

$$\Delta T = T - T_{ref}$$

The thermal strains are positive if $T>T_{ref}$. The *actual* strain diagram is a line since we assumed that plane sections remain plane. Therefore, the actual strains take the form:

$$\varepsilon = \varepsilon_0 + K \cdot x$$

The *elastic* strains are:

$$\varepsilon_\sigma = \varepsilon_f - \varepsilon = \varepsilon_f - (\varepsilon_0 + K \cdot x)$$

Therefore, the forces for the concrete blocks are:

$$dF_{ci} = E_c \cdot \varepsilon_{\sigma i} \cdot dA_{ci} = E_c \cdot \left(\varepsilon_{fi} - \left(\varepsilon_0 + K \cdot x_{ci}\right)\right) \cdot dA_{ci}$$

Where i=1..N, x_{ci} is the distance of the centroid of the concrete block from the top fibre. Similarly, the forces for the steel elements are:

$$dF_{sj} = E_s \cdot \varepsilon_{\sigma j} \cdot dA_{sj} = E_s \cdot \left(\varepsilon_{fj} - \left(\varepsilon_0 + K \cdot x_{sj}\right)\right) \cdot dA_{sj}$$

Where j=1..R, x_{sj} is the distance of the centroid of the steel element from the top fibre. Force equilibrium yields:

$$0 = \sum_{i=1}^{N} dF_{ci} + \sum_{j=1}^{R} dF_{sj}$$

$$0 = E_c \cdot \sum_{i=1}^{N} \left(\varepsilon_{fi} - \left(\varepsilon_0 + K \cdot x_{ci}\right)\right) \cdot dA_{ci} + E_s \cdot \sum_{j=1}^{R} \left(\varepsilon_{fj} - \left(\varepsilon_0 + K \cdot x_{sj}\right)\right) \cdot dA_{sj}$$

$$\sum_{i=1}^{N} \varepsilon_{fi} \cdot dA_{ci} + m \cdot \sum_{j=1}^{R} \varepsilon_{fj} \cdot dA_{sj} =$$
$$= \varepsilon_0 \cdot \left(\sum_{i=1}^{N} dA_{ci} + m \cdot \sum_{j=1}^{R} dA_{sj} \right) + K \cdot \left(\sum_{i=1}^{N} x_{ci} \cdot dA_{ci} + m \cdot \sum_{j=1}^{R} x_{sj} \cdot dA_{sj} \right)$$

Where:

$$m = \frac{E_s}{E_c}$$

But:

$$A_0 = \sum_{i=1}^{N} dA_{ci} + m \cdot \sum_{j=1}^{R} dA_{sj}$$

A_0 is the equivalent concrete area, and:

$$A_1 = \sum_{i=1}^{N} x_{ci} \cdot dA_{ci} + m \cdot \sum_{j=1}^{R} x_{sj} \cdot dA_{sj}$$

A_1 is the equivalent first moment of area around the top fibre. Therefore, the first equilibrium equation takes the form:

$$\sum_{i=1}^{N} \varepsilon_{fi} \cdot dA_{ci} + m \cdot \sum_{j=1}^{R} \varepsilon_{fj} \cdot dA_{sj} = \varepsilon_0 \cdot A_0 + K \cdot A_1 \quad (4.2)$$

Moment equilibrium yields a second equation in ε_0 and K. The moments (around the top fibre) from the concrete blocks are:

$$dM_{ci} = E_c \cdot \varepsilon_{ci} \cdot x_{ci} \cdot dA_{ci} = E_c \cdot \left(\varepsilon_{fi} - \left(\varepsilon_0 + K \cdot x_{ci}\right)\right) \cdot x_{ci} \cdot dA_{ci}$$

Similarly, for the steel elements:

$$dM_{sj} = E_s \cdot \varepsilon_{sj} \cdot x_{sj} \cdot dA_{sj} = E_s \cdot \left(\varepsilon_{fj} - \left(\varepsilon_0 + K \cdot x_{sj}\right)\right) \cdot x_{sj} \cdot dA_{sj}$$

Then for equilibrium:

$$M = \sum_{i=1}^{N} dM_{ci} + \sum_{j=1}^{R} dM_{sj}$$

Where M is the applied bending moment at the section. The above equation becomes:

$$M = E_c \cdot \sum_{i=1}^{N} \left(\varepsilon_{fi} - \left(\varepsilon_0 + K \cdot x_{ci}\right)\right) \cdot x_{ci} \cdot dA_{ci} + E_s \cdot \sum_{j=1}^{R} \left(\varepsilon_{fj} - \left(\varepsilon_0 + K \cdot x_{sj}\right)\right) \cdot x_{sj} \cdot dA_{sj}$$

$$\sum_{i=1}^{N} \varepsilon_{fi} \cdot x_{ci} \cdot dA_{ci} + m \cdot \sum_{j=1}^{R} \varepsilon_{fj} \cdot x_{sj} \cdot dA_{sj} =$$

$$= \varepsilon_0 \cdot \left(\sum_{i=1}^{N} x_{ci} \cdot dA_{ci} + m \cdot \sum_{j=1}^{R} x_{sj} \cdot dA_{sj} \right) + K \cdot \left(\sum_{i=1}^{N} x_{ci}^2 \cdot dA_{ci} + m \cdot \sum_{j=1}^{R} x_{sj}^2 \cdot dA_{sj} \right) + \frac{M}{E_c}$$

Or:

$$\sum_{i=1}^{N} \varepsilon_{fi} \cdot x_{ci} \cdot dA_{ci} + m \cdot \sum_{j=1}^{R} \varepsilon_{fj} \cdot x_{sj} \cdot dA_{sj} = \varepsilon_0 \cdot A_1 + K \cdot A_2 + \frac{M}{E_c} \quad (4.3)$$

Where:

$$A_2 = \sum_{i=1}^{N} x_{ci}^2 \cdot dA_{ci} + m \cdot \sum_{j=1}^{R} x_{sj}^2 \cdot dA_{sj}$$

A_2 is the equivalent second moment of area around the top fibre.

Finally, we can solve this 2 x 2 system of equations in ε_0 and K:

$$\begin{bmatrix} \varepsilon_0 \\ K \end{bmatrix} = \begin{bmatrix} A_0 & A_1 \\ A_1 & A_2 \end{bmatrix}^{-1} \cdot \left(\begin{bmatrix} \sum_{i=1}^{N} \varepsilon_{fi} \cdot dA_{ci} + m \cdot \sum_{j=1}^{R} \varepsilon_{sj} \cdot dA_{sj} \\ \sum_{i=1}^{N} \varepsilon_{fi} \cdot x_{ci} \cdot dA_{ci} + m \cdot \sum_{j=1}^{R} \varepsilon_{sj} \cdot x_{sj} \cdot dA_{sj} \end{bmatrix} - \begin{bmatrix} 0 \\ \frac{M}{E_c} \end{bmatrix} \right) \quad (4.4)$$

For given ε_{fi} and ε_{fj}, we can determine ε_0 and K using (4.4). The stresses of the concrete blocks and steel elements are then evaluated from ε_0 and K:

$$\sigma_{ci} = E_c \cdot \varepsilon_{\sigma i} = E_c \cdot \left(\varepsilon_{fi} - \left(\varepsilon_0 + K \cdot x_{ci} \right) \right) \quad (4.5)$$

$$\sigma_{sj} = E_s \cdot \varepsilon_{\sigma j} = E_s \cdot \left(\varepsilon_{fj} - \left(\varepsilon_0 + K \cdot x_{sj} \right) \right) \quad (4.6)$$

Using this approach, we can determine:

- The **elastic solution**, by setting all ε_{fi} equal to zero. In order to take prestress into account, we must introduce *initial* strains in the steel such that they generate the required amount of prestress in the concrete. As a first approximation:

$$\varepsilon_{sj} = \frac{F}{A_{sj} \cdot E_s}$$

  We probably need to update the above value because of the elastic deformations which change the prestress in concrete.

- The **thermo – elastic solution**, by setting all ε_{fi}, ε_{fj} equal to the corresponding $\Delta\varepsilon_{\alpha i}$, $\Delta\varepsilon_{\alpha j}$ i.e. the thermal strains, as defined in (4.1). The prestress is taken into account as previously.

- The **creep solution**. The process is the following:

  i. Divide the time span into small increments of Δt^*, i.e. pseudo – time.

  ii. Initially, set the free strains ε_{fi}, ε_{fj} equal to the corresponding $\Delta\varepsilon_{\alpha i}$, $\Delta\varepsilon_{\alpha j}$ i.e. the thermal strains, as defined in (4.1). The prestress is taken into account as previously.

  iii. As a repeated process, i.e. a loop, do the following:

    a. Calculate ε_0 and K.

    b. Calculate the stress of each element, i.e. concrete blocks and steel elements.

    c. Assuming that the stresses remain constant during Δt^*, calculate the concrete creep strains that develop during Δt^*:

$$\Delta\varepsilon_{ci} = \sigma_{ci} \cdot \Phi(T_{ci}) \cdot \Delta t^* \quad (4.7)$$

    There are no creep strains for steel. However, it may be necessary to allow for relaxation.

    d. Calculate the *updated* free strains:

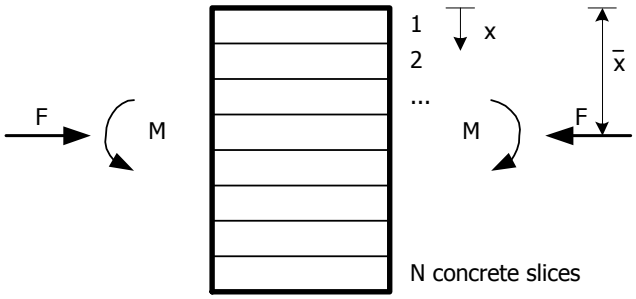
$$\varepsilon_{fi}^{new} = \varepsilon_{fi} - \Delta\varepsilon_{ci} \quad (4.8)$$

    e. Repeat the loop.

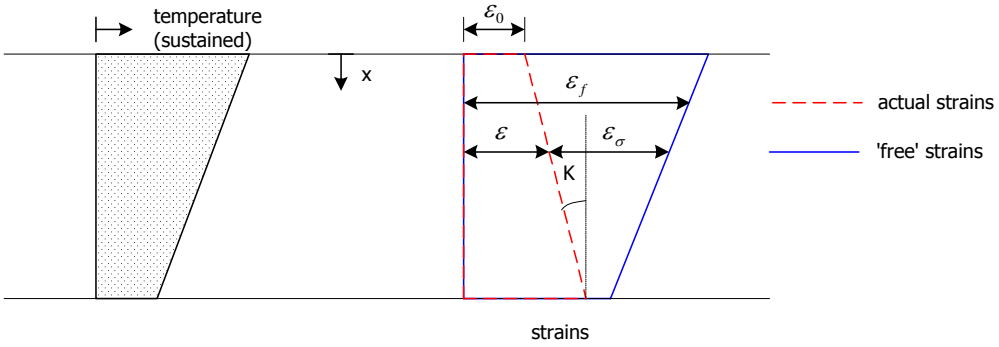
In this example all creep analysis takes place in *pseudo – time*. This is for convenience since the temperature crossfall is sustained. Also, it is obvious that work is simplified when the analysis is related directly to the normalised creep parameter (pseudo – time) and real time is eliminated so far as the calculations are concerned; whether the stresses are in stationary state, i.e. whether creep is completed, is governed by pseudo – time *alone*. We can

associate pseudo – time with real time by using the normalised creep diagram of the specific concrete, as it was defined earlier. In other cases, creep analysis can take place in real time.

➢ **Analysis (b):** Both the bending moment and the axial force are specified at all times. This implies that the axial force is a *follow – up* load.



Analysis (b)



In this case, force equilibrium yields:

$$F = \sum_{i=1}^{N} dF_{ci}$$

$$F = E_c \cdot \sum_{i=1}^{N} \left(\varepsilon_{fi} - \left(\varepsilon_0 + K \cdot x_{ci} \right) \right) \cdot dA_{ci}$$

$$\sum_{i=1}^{N} \varepsilon_{fi} \cdot dA_{ci} = \varepsilon_0 \cdot \left(\sum_{i=1}^{N} dA_{ci} \right) + K \cdot \left(\sum_{i=1}^{N} x_{ci} \cdot dA_{ci} \right) + \frac{F}{E_c}$$

Similarly:

$$A_0 = \sum_{i=1}^{N} dA_{ci}$$

$$A_1 = \sum_{i=1}^{N} x_{ci} \cdot dA_{ci}$$

Therefore, the first equilibrium equation takes the form:

$$\sum_{i=1}^{N} \varepsilon_{fi} \cdot dA_{ci} = \varepsilon_0 \cdot A_0 + K \cdot A_1 + \frac{F}{E_c} \quad (4.9)$$

Moment equilibrium yields a second equation in ε_0 and K:

$$M^* = \sum_{i=1}^{N} dM_{ci}$$

Where M^* is the resulting bending moment around the top fibre of the section:

$$M^* = M + F \cdot \bar{x}$$

Therefore:

$$M^* = E_c \cdot \sum_{i=1}^{N} \left(\varepsilon_{fi} - \left(\varepsilon_0 + K \cdot x_{ci} \right) \right) \cdot x_{ci} \cdot dA_{ci}$$

$$\sum_{i=1}^{N} \varepsilon_{fi} \cdot x_{ci} \cdot dA_{ci} = \varepsilon_0 \cdot \left(\sum_{i=1}^{N} x_{ci} \cdot dA_{ci} \right) + K \cdot \left(\sum_{i=1}^{N} x_{ci}^2 \cdot dA_{ci} \right) + \frac{M^*}{E_c}$$

Or:

$$\sum_{i=1}^{N} \varepsilon_{fi} \cdot x_{ci} \cdot dA_{ci} = \varepsilon_0 \cdot A_1 + K \cdot A_2 + \frac{M^*}{E_c} \quad (4.10)$$

Where:

$$A_2 = \sum_{i=1}^{N} x_{ci}^2 \cdot dA_{ci}$$

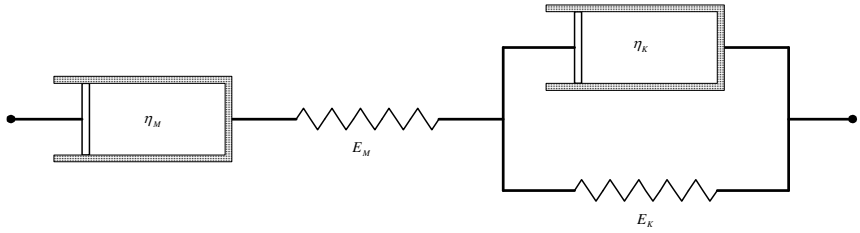
Finally, we can solve this 2 x 2 system of equations in ε_0 and K:

$$\begin{bmatrix} \varepsilon_0 \\ K \end{bmatrix} = \begin{bmatrix} A_0 & A_1 \\ A_1 & A_2 \end{bmatrix}^{-1} \cdot \left(\begin{bmatrix} \sum_{i=1}^{N} \varepsilon_{fi} \cdot dA_{ci} \\ \sum_{i=1}^{N} \varepsilon_{fi} \cdot x_{ci} \cdot dA_{ci} \end{bmatrix} - \begin{bmatrix} \frac{F}{E_c} \\ \frac{M^*}{E_c} \end{bmatrix} \right) \quad (4.11)$$

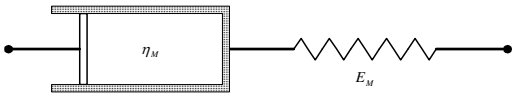
The rest of the creep analysis is the same as (a).

4.1.3 Rate of Flow

This creep law is based on Burgars body, which is a combination of a Maxwell and a Kelvin element connected in series:



For the Maxwell element:

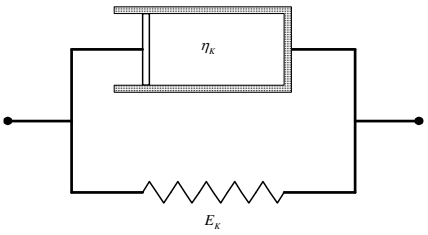


$$\varepsilon_M = \frac{\sigma}{E_M} + \int \frac{\sigma}{\eta_M} dt$$

At *constant stress*:

$$\dot{\varepsilon}_M = \frac{\sigma}{\eta_M}$$

Where the dot notation refers to differentiation with respect to time. For the Kelvin element:



$$\sigma = \sigma_1 + \sigma_2$$

$$\varepsilon_K = \frac{\sigma_1}{E_K}$$

$$\dot{\varepsilon}_K = \frac{\sigma_2}{\eta_K}$$

Therefore:

$$\sigma = \varepsilon_K \cdot E_K + \dot{\varepsilon}_K \cdot \eta_K$$

At *constant stress*, the solution is:

$$\varepsilon_K = c \cdot e^{-\frac{E_K}{\eta_K} \cdot t} + \frac{\eta_K}{E_K} \cdot \frac{\sigma}{\eta_K}$$

At t=0 $\Rightarrow$ ε_K=0, therefore:

$$c = -\frac{\sigma}{E_K}$$

$$\varepsilon_K = -\frac{\sigma}{E_K} \cdot e^{-\frac{E_K}{\eta_K} \cdot t} + \frac{\sigma}{E_K}$$

Or:

$$\varepsilon_K = \frac{\sigma}{E_K} \cdot \left(1 - e^{-\frac{E_K}{\eta_K} \cdot t} \right)$$

Also:

$$\dot{\varepsilon}_K = \frac{\sigma}{\eta_K} \cdot e^{-\frac{E_K}{\eta_K} \cdot t}$$

Noting that:

$$\varepsilon_K(\infty) = \frac{\sigma}{E_K}$$

We can write:

$$\varepsilon_K(\infty) - \varepsilon_K(t) = \dot{\varepsilon}_K \cdot \frac{\eta_K}{E_K}$$

Or:

$$\dot{\varepsilon}_K = (\varepsilon_K(\infty) - \varepsilon_K(t)) \cdot \frac{E_K}{\eta_K}$$

For Burgars body, the total strain rate due to stress σ is:

$$\dot{\varepsilon} = \dot{\varepsilon}_M + \dot{\varepsilon}_K$$

$$\dot{\varepsilon} = \frac{\sigma}{\eta_M} + (\varepsilon_K(\infty) - \varepsilon_K(t)) \cdot \frac{E_K}{\eta_K} \quad (4.12)$$

In the step – by – step analysis, the stress during each time interval is assumed to be constant. The resulting creep strain increment is then:

$$\Delta\varepsilon_c = \dot{\varepsilon} \cdot \Delta t \quad (4.13)$$

Where the strain rate is given from (4.12). Note that $\varepsilon_K(\infty)$ is always known from the current stress state, $\varepsilon_K(t)$ is known from the accumulation of $\Delta\varepsilon_K$ over all previous intervals, while η_K, η_M and E_K are defined material constants.

4.2 Semi – analytical

4.2.1 General considerations

Semi – analytical creep analyses are approximate time – dependent solutions which invoke the principle of Virtual Power.

These analyses can accommodate the following creep laws:

- Rate of Creep
- Rate of Flow

As the main part of this dissertation is based on numerical step – by – step analysis, the semi – analytical creep analyses will not be presented in detail.

4.2.2 Rate Of Creep

For a structure in which internal strains and external displacements exhibit time – dependent variations, the principle of Virtual Power may be invoked:

$$\int_{volume} \dot{\varepsilon} \cdot \delta\sigma \, dV - \int_{surface} \dot{u}_s \cdot \delta R_s ds = 0 \quad (4.14)$$

Where $\dot{\varepsilon}$, $\dot{u}_s$ are respectively the actual, and therefore compatible, internal strain rates and displacement rates at the supports, while $\delta\sigma$ and δR_s consist of an equilibrium set of internal stresses and support reactions that are independent of any real loading on the structure.

In reduced form, i.e. no displacement rates at the supports, this principle states:

$$\int \dot{\varepsilon} \cdot \delta\sigma \, dV = 0 \quad (4.15)$$

This equation may be used to obtain a creep solution when strain rates are related to the actual stresses through an appropriate constitutive law, i.e. Rate of Creep (Maxwell) law.

The Maxwell creep law is:

$$\dot{\varepsilon} = \frac{\dot{\sigma}}{E} + \sigma \cdot \Phi(T)$$

The stresses take the following form:

$$\sigma = \sigma_0 + \sum_{i=1}^{N} a_i \cdot \sigma_i \quad (4.16)$$

Here σ_0 satisfies the boundary loading, σ_i represent sets of self – equilibrating stresses while α_i are time – dependent weighting parameters. σ_0 may be chosen to be the actual or thermoelastic solution at zero time, in which case $\alpha_i=0$ at zero time.

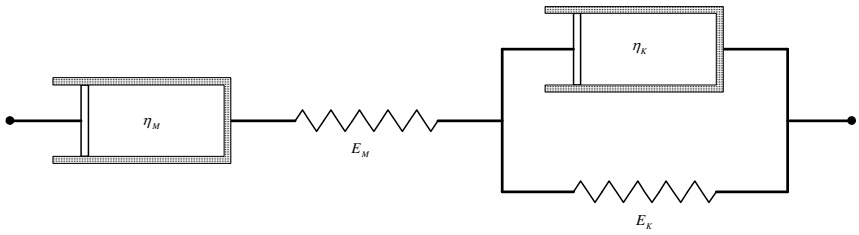
The use of Virtual Power theory, i.e. equation (4.15), combined with Maxwell creep law for each of the self – equilibrating sets of stresses leads to a system of *N* first order differential equations in α_i:

$$\underset{N\times N}{[A]} \cdot \underset{N\times 1}{\{\dot{a}\}} + \underset{N\times N}{[B]} \cdot \underset{N\times 1}{\{a\}} + \underset{N\times 1}{[C]} = \underset{N\times 1}{[0]} \quad (4.17)$$

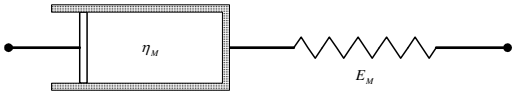
Where N is the number of self – equilibrating sets of stresses invoked in the analysis. By solving this system of equations, numerically if necessary, we obtain the variation of α_i with time. Therefore, using (4.16), we obtain the time – dependent stresses.

4.2.3 Rate Of Flow

This creep law is based on Burgars body, which is a combination of a Maxwell and a Kelvin element connected in series:



For the Maxwell element:

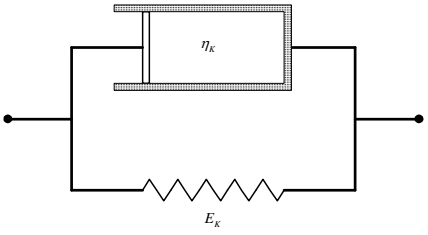


$$\varepsilon_M = \frac{\sigma}{E_M} + \int \frac{\sigma}{\eta_M} dt \Rightarrow$$

$$\dot{\varepsilon}_M = \frac{\dot{\sigma}}{E_M} + \frac{\sigma}{\eta_M} \Rightarrow$$

$$\ddot{\varepsilon}_M = \frac{\ddot{\sigma}}{E_M} + \frac{\dot{\sigma}}{\eta_M}$$

Where the dot notation refers to differentiation with respect to time. For the Kelvin element:



$$\sigma = \sigma_1 + \sigma_2$$

$$\varepsilon_K = \frac{\sigma_1}{E_K}$$

$$\dot{\varepsilon}_K = \frac{\sigma_2}{\eta_K}$$

Therefore:

$$\sigma = \varepsilon_K \cdot E_K + \dot{\varepsilon}_K \cdot \eta_K \Rightarrow$$

$$\dot{\sigma} = \dot{\varepsilon}_K \cdot E_K + \ddot{\varepsilon}_K \cdot \eta_K$$

For Burgars body, the total strains are:

$$\varepsilon = \varepsilon_M + \varepsilon_K$$

Combining the two elements, we get:

$$\ddot{\varepsilon}_K \cdot \eta_K + \dot{\varepsilon}_K \cdot E_K = \dot{\sigma}$$

$$\ddot{\varepsilon}_M \cdot \eta_K + \dot{\varepsilon}_M \cdot E_K = \eta_K \cdot \left(\frac{\ddot{\sigma}}{E_M} + \frac{\dot{\sigma}}{\eta_M} \right) + E_K \cdot \left(\frac{\dot{\sigma}}{E_M} + \frac{\sigma}{\eta_M} \right)$$

By adding the above two equations, we get:

$$\ddot{\varepsilon} \cdot \eta_K + \dot{\varepsilon} \cdot E_K = \eta_K \cdot \left(\frac{\ddot{\sigma}}{E_M} + \frac{\dot{\sigma}}{\eta_M} \right) + E_K \cdot \left(\frac{\dot{\sigma}}{E_M} + \frac{\sigma}{\eta_M} \right) + \dot{\sigma} \Rightarrow$$

$$\ddot{\varepsilon} \cdot \eta_K + \dot{\varepsilon} \cdot E_K = \frac{\eta_K}{E_M} \cdot \ddot{\sigma} + \left(1 + \frac{\eta_K}{\eta_M} + \frac{E_K}{E_M} \right) \cdot \dot{\sigma} + \frac{E_K}{\eta_M} \cdot \sigma$$

Or:

$$\frac{\eta_K}{E_K} \cdot \ddot{\varepsilon} + \dot{\varepsilon} = \frac{\eta_K}{E_M \cdot E_K} \cdot \ddot{\sigma} + \left(\frac{1}{E_K} + \frac{\eta_K}{\eta_M \cdot E_K} + \frac{1}{E_M} \right) \dot{\sigma} + \frac{1}{\eta_M} \cdot \sigma \quad (4.18)$$

The principle of Virtual Power may be invoked:

$$\int \varepsilon \cdot \delta\sigma \, dV = 0 \text{ (Virtual Work)}$$

$$\int \dot{\varepsilon} \cdot \delta\sigma \, dV = 0 \text{ (Virtual Power)}$$

$$\int \ddot{\varepsilon} \cdot \delta\sigma \, dV = 0 \text{ (Etc)}$$

As long as η_K and E_K don't vary within the structure, we can write:

$$\int \frac{\eta_K}{E_K} \cdot \ddot{\varepsilon} \cdot \delta\sigma \, dV + \int \dot{\varepsilon} \cdot \delta\sigma \, dV = \frac{\eta_K}{E_K} \cdot \int \ddot{\varepsilon} \cdot \delta\sigma \, dV + \int \dot{\varepsilon} \cdot \delta\sigma \, dV = 0 \quad (4.19)$$

As long as (4.19) holds, we can proceed and select self – equilibrating sets of stresses. The stresses take the following form:

$$\sigma = \sigma_0 + \sum_{i=1}^{N} a_i \cdot \sigma_i \quad (4.20)$$

Here σ_0 satisfies the boundary loading, σ_i represent sets of self – equilibrating stresses while α_i are time – dependent weighting parameters. σ_0 may be chosen to be the actual or thermoelastic solution at zero time, in which case α_i=0 at zero time.

The use of Virtual Power theory, i.e. equation (4.19), combined with (4.18) for each of the self – equilibrating sets of stresses leads to a system of *N* second order differential equations in α_i:

$$\underset{N\times N}{[A]} \cdot \underset{N\times 1}{\{\ddot{a}\}} + \underset{N\times N}{[B]} \cdot \underset{N\times 1}{\{\dot{a}\}} + \underset{N\times N}{[C]} \cdot \underset{N\times 1}{\{a\}} + \underset{N\times 1}{[D]} = \underset{N\times 1}{[0]} \quad (4.21)$$

Where N is the number of self – equilibrating sets of stresses invoked in the analysis. By solving this system of equations, numerically if necessary, we either obtain the fractions of time for each α_i term or else values of the α_i terms at discrete instances of time.

5. Short And Long Term Deformation And Stressing Of Slender Cylindrical Concrete Piers Due To Solar Heating

5.1 Brief

We will investigate the effect of non – uniform solar heating of slender cylindrical concrete piers based on a numerical step – by – step creep analysis and Maxwell creep law. As the basics of this analysis have already been presented in the previous chapter, the reader is urged to study paragraph 4.1.2 first. We will obtain both short and long – term solutions i.e. thermoelastic and creep solutions. The analysis will be carried out with a user – friendly computer program which was developed as part of this dissertation. Details about the computer program can be found in the next chapter.

5.2 General assumptions

As a first approach, we could assume that solar heating is a cyclic temperature variation. However, the movement of the sun is not the same throughout the year and the temperatures vary significantly e.g. from winter to summer. In order to obtain realistic results we have to adopt a more robust approach, i.e. take into account the time of the year and the climatic conditions of the specific area where the pier is located.

We will assume that the pier is located on the north hemisphere. Therefore, the sun rises from the East, moves to the South during the day, and finally sets West.

The pier will be treated as a cantilever with a compressive force on top. This force is a *follow up load*; therefore analysis (b) will be invoked. This analysis was presented for a rectangular cross section in paragraph 4.1.2.

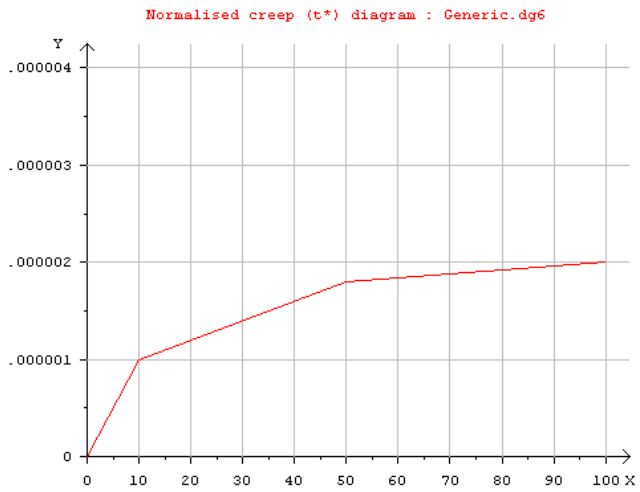
The structure will be divided into *rings*, *sectors* and *slices*. We assume that each block of concrete has *uniform* temperature at all times; this value of temperature, of course, varies in time. Moreover, we assume that the temperature of each block is governed by the position of *the centroid* of the block only; therefore, in order to calculate the temperature of each block at each time, we need to determine the position of the centroid of the block in space and the relative position of the centroid with respect to the sun. It is obvious that as the number of blocks invoked in the analysis increases, the variation of temperature becomes smoother.

5.3 Data input

In order to deal with the complexity of the problem, there are two kinds of data which are fed into the computer program:

- **Variables:** These are *discrete variables* which either refer to the whole structure or are assumed to be common within the structure, e.g. the dimensions of the pier, the coefficient of thermal expansion, the reference temperature, the number of rings, sectors, slices, etc.

- **Diagrams:** The computer program also uses *diagrams* which provide complete freedom to the user. There are various diagrams invoked in the analysis, such as the position of the sun as a function of time, the ambient temperature as a function of time, etc. The use of each diagram is described in detail later.



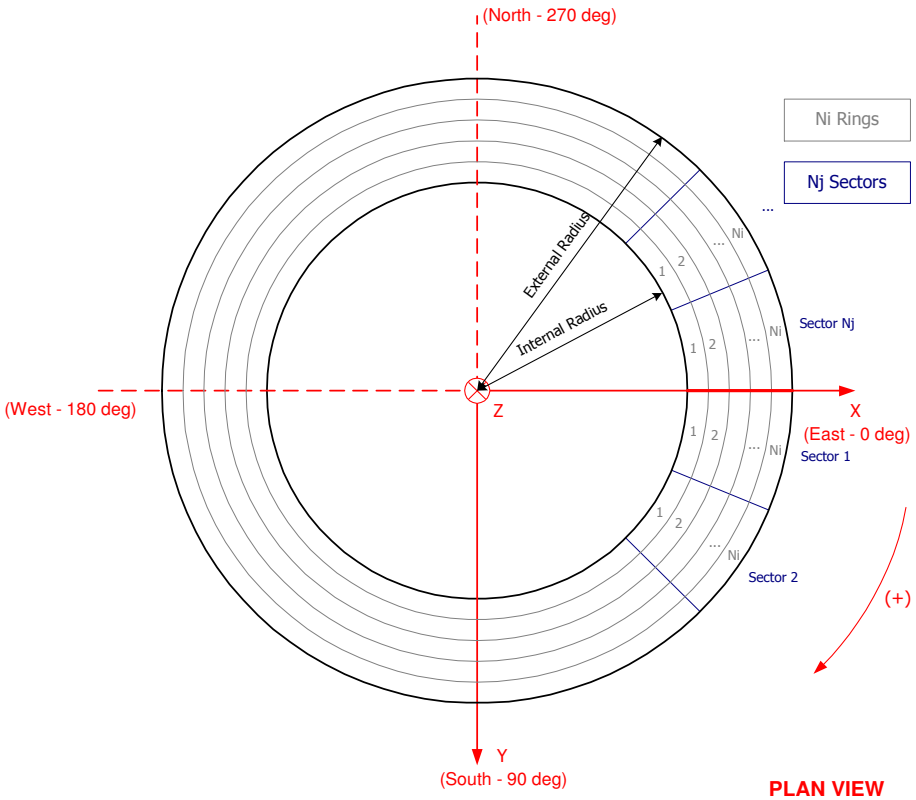
The diagrams are defined as a series of discrete points, i.e. pairs of x – y values. The variation of the diagrams between adjacent discrete points is assumed to be *linear*; however, there are but practical limitations to the number of the points invoked in the definition of a diagram. Therefore, the diagram can be *as smooth as desired* with the use of more points. In order to calculate the value of a function for a specific value of x, linear interpolation is performed.

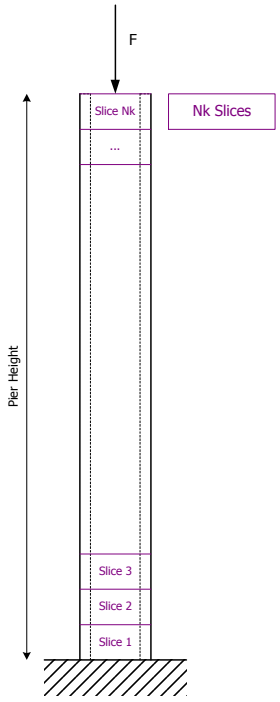
The points of a diagram can be edited within the computer program. As each diagram is stored as an individual file, *a library of diagrams* can be constructed based on *experimental* data of *real* structures. Moreover, as far as the creep analysis is concerned, the computer program allows the assignment of *different* diagrams *for each month of the year*.

5.4 Discretization of the structure

We will now proceed to the discretization of the structure. The origin of the XYZ Cartesian coordinate system is located at the tip of the pier; X – axis extends to the East (0^0), Y – axis extends to the South (90^0) and Z – axis

extends to the centre of the earth. The structure is divided into N_i rings, N_j sectors and N_k slices, as shown in the figures:

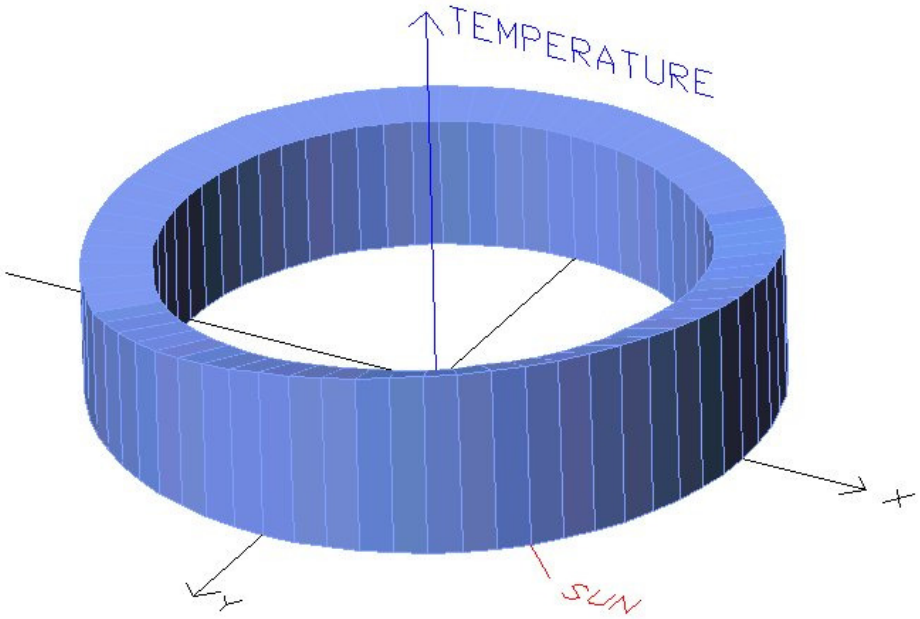


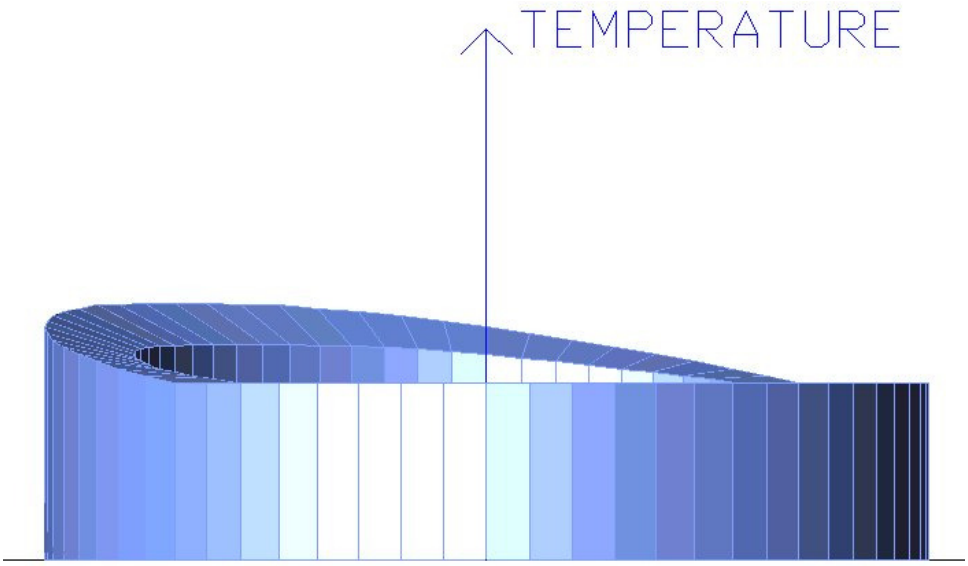


5.5 Temperature calculations

5.5.1 General considerations

It is obvious that the part of the pier which faces the sun is hotter than the environment. The actual variation of temperature within a cross section because of solar heating might look like this:



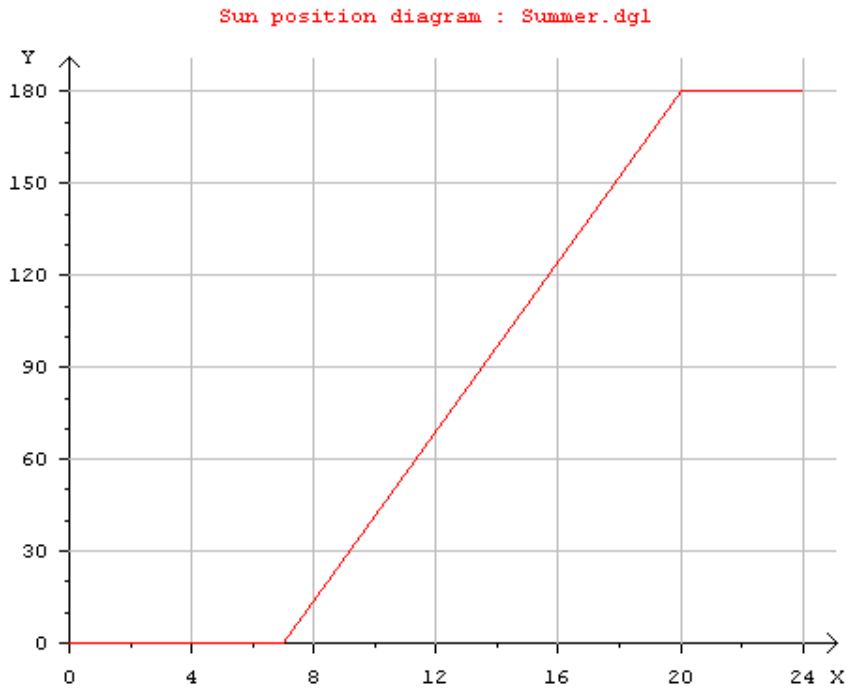


In order to reduce the computational effort, we can make the reasonable assumption that the temperature varies only within the cross section and not in height. Therefore, each block has the same temperature with the block on top and the block below at all times.

The objective is to be able to calculate the temperature of each concrete block as a function of the position of the sun and ultimately as a function of time. We also need to be able to describe the temperature variations of the pictures. This was accomplished with the use of certain diagrams:

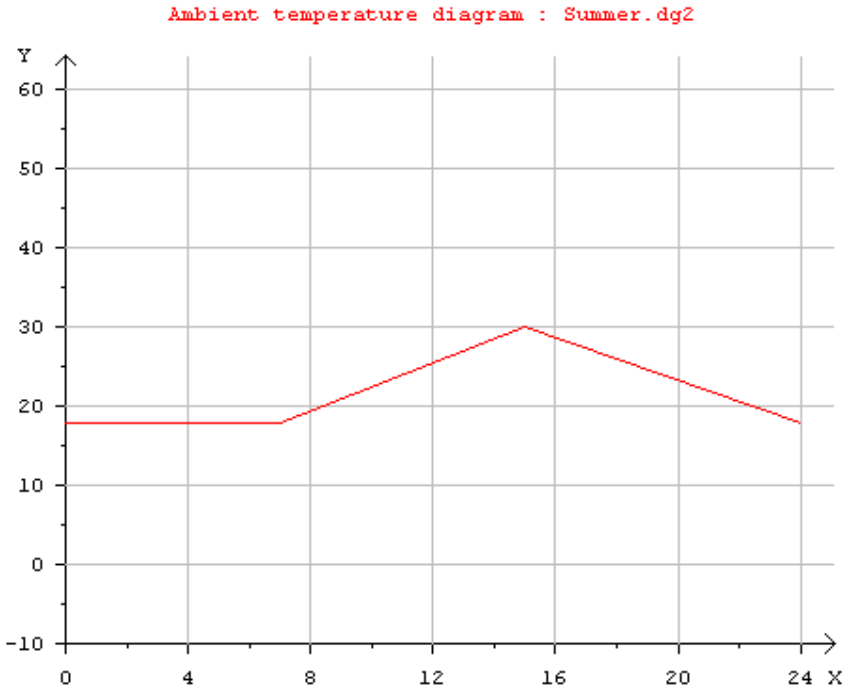
5.5.2 Diagram I: Sun position

The first diagram provides the position of the sun as a function of the time of the day. Therefore, X values vary in the range of 0 to 24 (decimal) hours whereas Y values vary in the range of 0 degrees (East) to 180 degrees (West), e.g.:



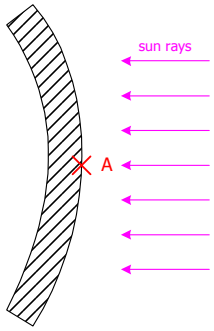
5.5.3 Diagram II: Ambient temperature

The second diagram provides the variation of the ambient temperature, i.e. the temperature of the environment, as a function of the time of the day. Therefore, X values vary in the range of 0 to 24 (decimal) hours, e.g.:



5.5.4 Diagram III: Surface temperature

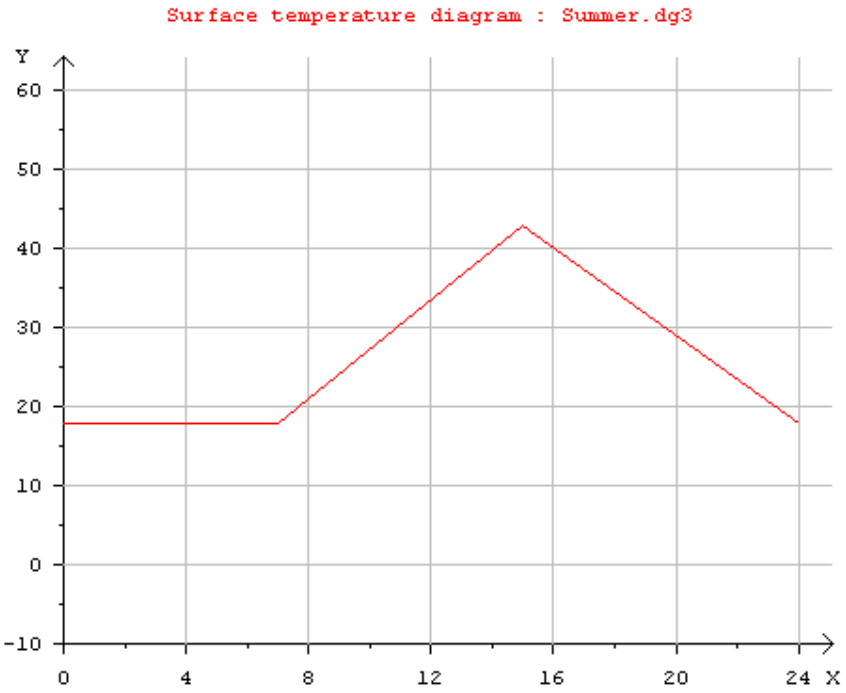
The third diagram provides the variation, during the day, of the temperature of a concrete surface which *directly* faces the sun, i.e. point A of the following figure:



It is obvious that the surface temperature must be greater than or equal to the ambient temperature *at all times*. The two temperatures should be equal in the following cases:

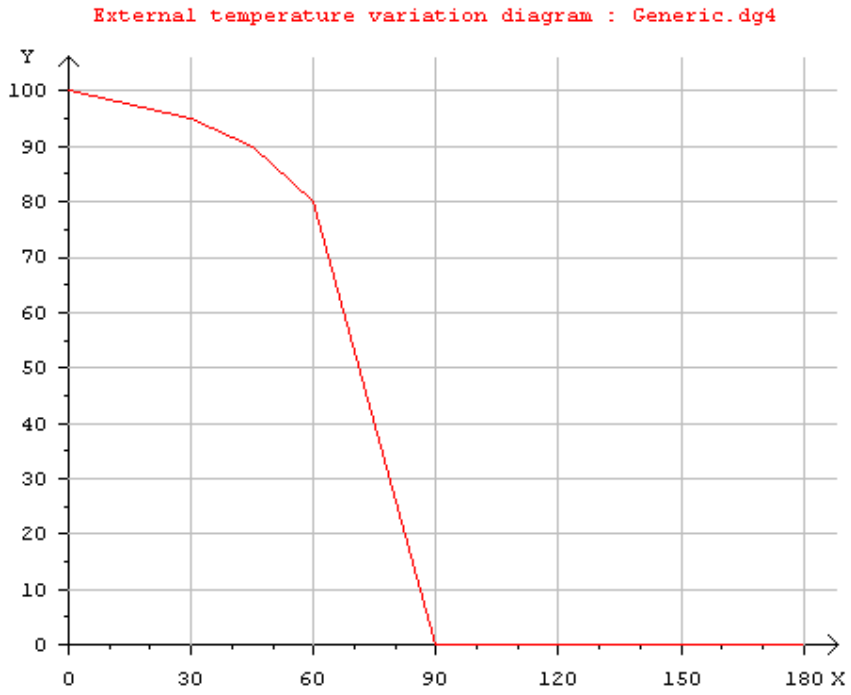
- During the night, where there is no sun.
- When there are many clouds in the sky, which prevent the sun rays from heating concrete above the ambient temperature.

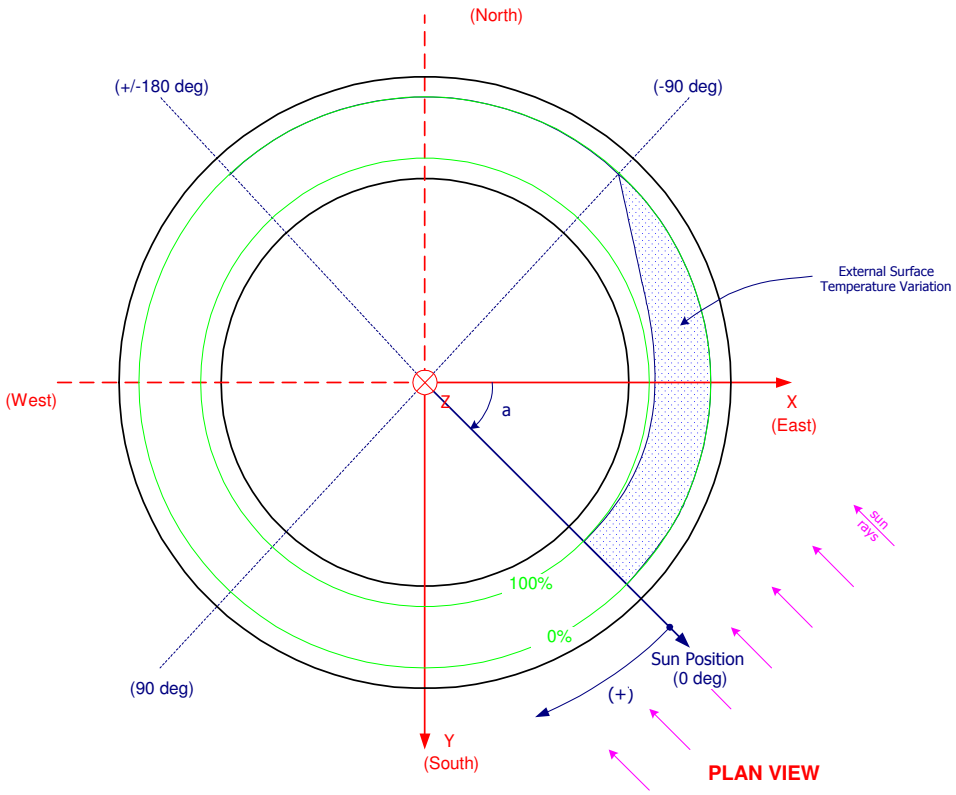
As previously, X values vary in the range of 0 to 24 (decimal) hours, e.g.:



5.5.5 Diagram IV: External surface temperature variation

The fourth diagram provides the variation between *the surface temperature* and *the ambient temperature* of the *external surface* of the pier as a function of the relative position of the concrete block with respect to the sun. X values vary in the range of 0 to 180 degrees. We assume that a symmetrical diagram applies in the range of 0 to -180 degrees; therefore, the diagram is sufficient for the whole perimeter of the pier. Y values vary in the range of 0 to 100:





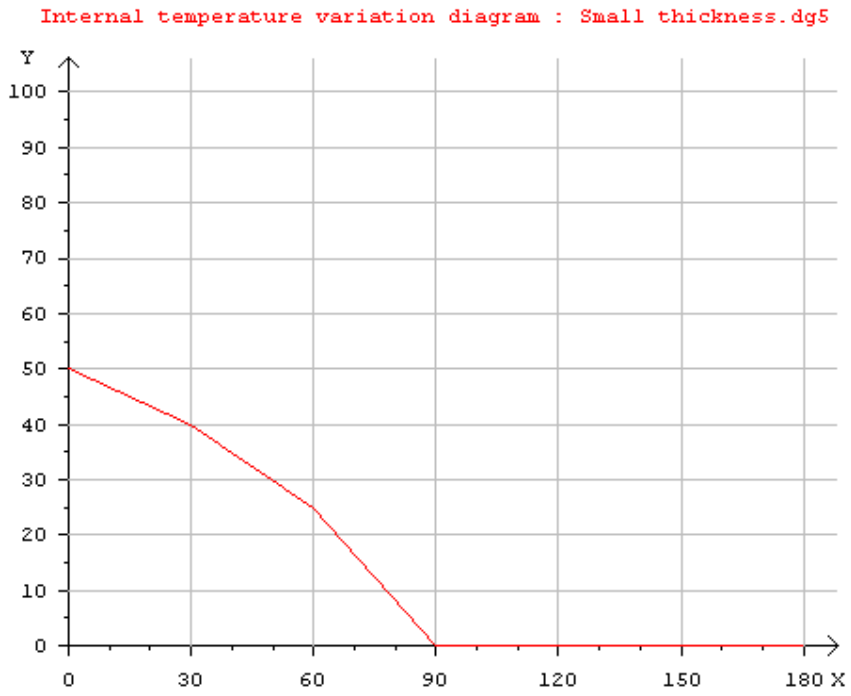
The origin of the diagram (0 degrees) refers *always* to the position of sun. Y values indicate the variation of the temperature of the external surface around the perimeter. The temperature varies between the ambient temperature T_a (0%) and the surface temperature T_s (100%) at the specific time of the calculation. Therefore:

$$T = T_a + (T_s - T_a) \cdot Y\%$$

It is obvious that for small values of X the values of Y should be close to 100, because this part of the external perimeter is heated intensively by the sun, i.e. the sun rays are almost normal to the surface. The values of Y reduce as the relative angle increases. Values of X in the range of 90 to 180 degrees refer to that part of the perimeter which doesn't face the sun; therefore, in this case, Y should be zero. In this way, the "dark" side of the pier follows the ambient temperature at all times.

5.5.6 Diagram V: Internal surface temperature variation

The fifth diagram is similar to the previous one, but it refers to the *internal* surface of the pier. As before, X values vary in the range of 0 to 180 degrees while Y values vary in the range of 0 to 100, e.g.:



5.5.7 Calculations

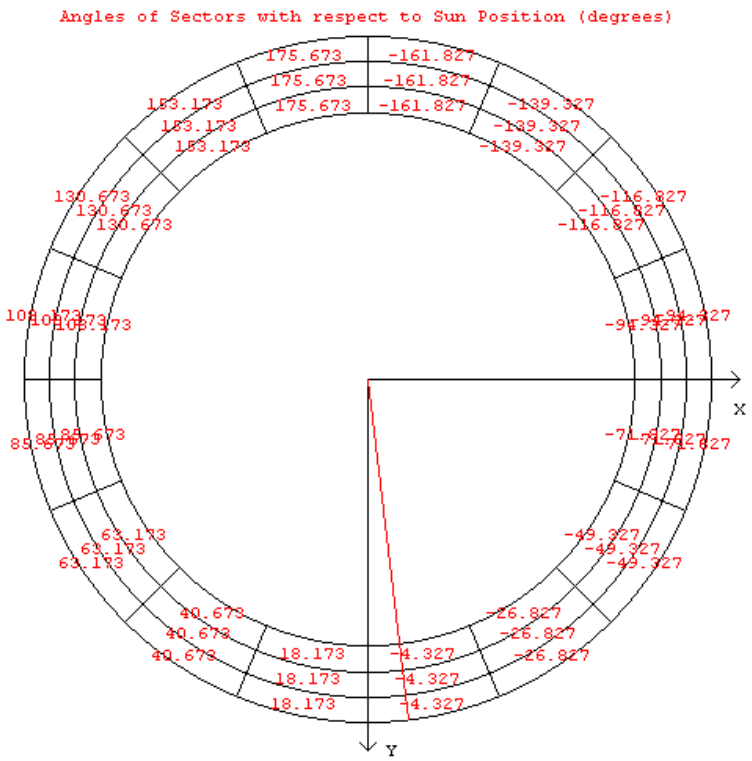
The process of calculating the temperature of each concrete block will be illustrated with an example. For clarity, only 3 rings and 16 sectors are invoked in the analysis:

- Pick the time of the day for the calculations, for example 13:00.
- Using diagram I, calculate the position of the sun with respect to East, for example a=83.076 degrees.

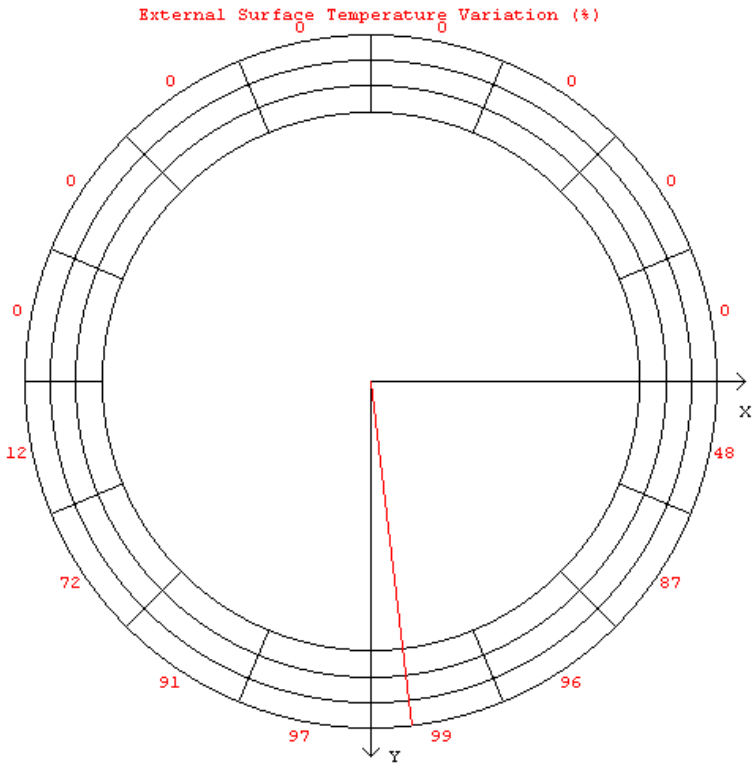
- Using diagram II, calculate the ambient temperature, for example T_a=27 degrees.

- Using diagram III, calculate the surface temperature, for example T_s=36.75 degrees.

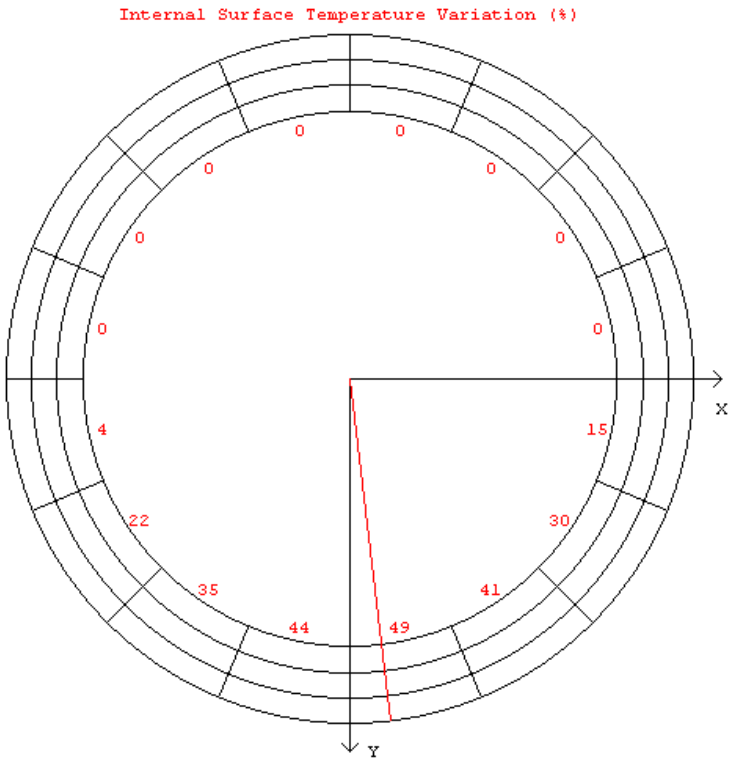
- Using diagram IV, calculate the temperature variation around the external surface of the pier. First we need to calculate the position of each sector with respect to the sun:



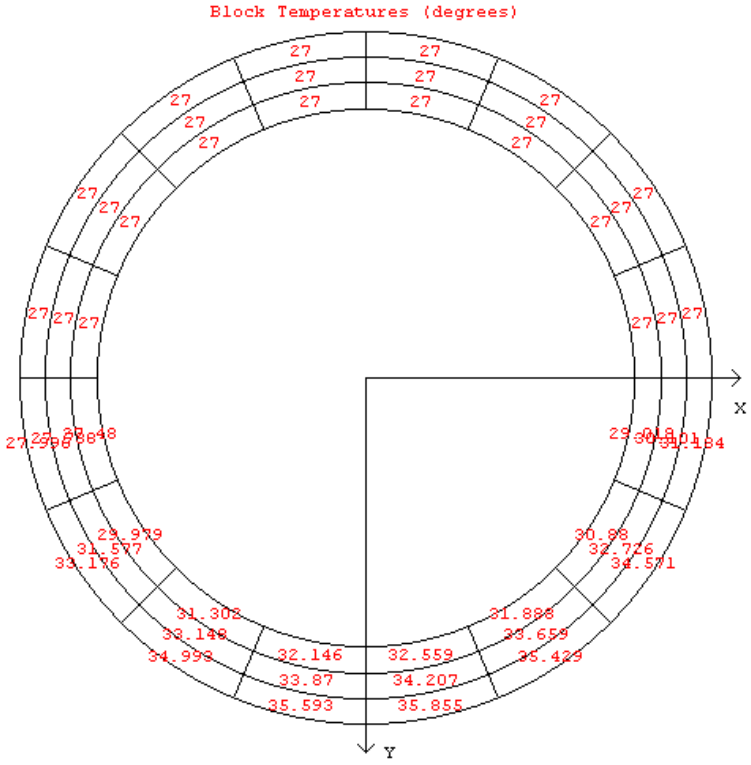
By applying these values to diagram IV, we obtain the external surface temperature variation:



- Similarly, calculate the temperature variation around the internal surface of the pier using diagram V:



- Assuming a *linear variation of temperatures along the thickness* of the pier, calculate the temperature of each concrete block. As mentioned earlier, the centroid of the block is the reference point:



5.5.8 Advantages

Using the above five diagrams, we have achieved the following:

- We have divided the description of a complex temperature variation into a number of parameters. These parameters are not abstract but relate to actual, and therefore measurable, quantities. Thus we have created a flexible, yet robust system that can describe complex temperature variations.

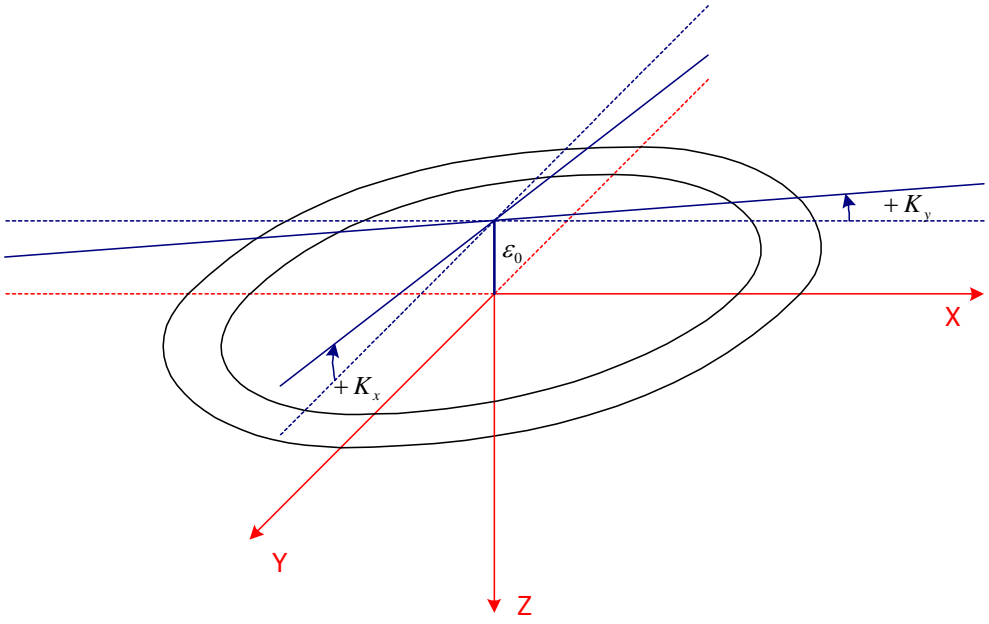
- The variation of temperature of *each* block is *smooth in time*. Therefore the temperature calculations are *independent of the time interval* invoked in the creep analysis.

The only assumptions of this analysis are:

- We assume a linear variation of temperatures along the thickness of the pier.
- The symmetry of the temperatures with respect to the position of the sun implies that the movement of the sun is *slow enough* for a symmetrical temperature variation to be established.

5.6 Equilibrium calculations

The *actual* strain variation forms a plane since we assumed that plane sections remain plane:



Therefore, the actual strains take the form:

$$\varepsilon = \varepsilon_0 + K_y \cdot x + K_x \cdot y \quad (5.1)$$

The *elastic* strains are:

$$\varepsilon_\sigma = \varepsilon_f - \varepsilon = \varepsilon_f - \left(\varepsilon_0 + K_y \cdot x + K_x \cdot y\right) \quad (5.2)$$

Where ε_f are the free strains. As previously, we can calculate the free *thermal* strains:

$$\varepsilon_{fa} = \overline{a} \cdot \Delta T \quad (5.3)$$

Where:

$$\Delta T = T - T_{ref} \quad (5.4)$$

The free thermal strains are positive if $T>T_{ref}$. In creep analysis, the free strains at each time are:

$$\varepsilon_f = \varepsilon_{fa} - \Delta\varepsilon_c \quad (5.5)$$

Where ε_{fa} are the free thermal strains at the same time and $\Delta\varepsilon_c$ are the creep strains that have been accumulated since the beginning of the analysis.

The forces for the concrete blocks are:

$$\begin{aligned} dF_{(i,j,k)} &= E_c \cdot \varepsilon_{\sigma(i,j,k)} \cdot dA_{(i)} = \\ &= E_c \cdot \left(\varepsilon_{f(i,j,k)} - \left(\varepsilon_{0(k)} + K_{y(k)} \cdot x_{(i,j)} + K_{x(k)} \cdot y_{(i,j)}\right)\right) \cdot dA_{(i)} \end{aligned} \quad (5.6)$$

Where the indexes reveal the scope in which each value varies, i.e. i for rings, j for sectors and k for slices.

Force equilibrium yields:

$$F_{(k)} = \sum_i \sum_j dF_{(i,j,k)}$$

Where $F_{(k)}$ is the axial compressive force for the k^{th} slice. This value is equal to the compressive force F for the topmost slice but increases towards the base of the pier because of the self – weight which is accumulated. Therefore:

$$\frac{F_{(k)}}{E_c} = \sum_i \sum_j \varepsilon_{f(i,j,k)} \cdot dA_{(i)} - \varepsilon_{0(k)} \cdot \sum_i \sum_j dA_{(i)} - \\ - K_{y(k)} \cdot \sum_i \sum_j x_{(i,j)} \cdot dA_{(i)} - K_{x(k)} \cdot \sum_i \sum_j y_{(i,j)} \cdot dA_{(i)} \quad (5.7)$$

Where:

$$\sum_i = \sum_{i=1}^{N_i} , \sum_j = \sum_{j=1}^{N_j} , \sum_k = \sum_{k=1}^{N_k}$$

We assume that:

$$\sum_i \sum_j x_{(i,j)} \cdot dA_{(i)} = \sum_i \sum_j y_{(i,j)} \cdot dA_{(i)} = 0 \quad (5.8)$$

Justification for the above assumption is given later in this paragraph. Therefore, (5.7) becomes:

$$\varepsilon_{0(k)} = \left(\sum_i \sum_j \varepsilon_{f(i,j,k)} \cdot dA_{(i)} - \frac{F_{(k)}}{E_c} \right) \cdot \frac{1}{\sum_i \sum_j dA_{(i)}} \quad (5.9)$$

Moment equilibrium around the X – axis yields:

$$0 = \sum_i \sum_j y_{(i,j)} \cdot dF_{(i,j,k)} \Rightarrow$$

$$0 = \sum_i \sum_j y_{(i,j)} \cdot \varepsilon_{f(i,j,k)} \cdot dA_{(i)} - \varepsilon_{0(k)} \cdot \sum_i \sum_j y_{(i,j)} \cdot dA_{(i)} - \\ - K_{y(k)} \cdot \sum_i \sum_j y_{(i,j)} \cdot x_{(i,j)} \cdot dA_{(i)} - K_{x(k)} \cdot \sum_i \sum_j y_{(i,j)}^2 \cdot dA_{(i)} \quad (5.10)$$

We also assume that:

$$\sum_i \sum_j x_{(i,j)} \cdot y_{(i,j)} \cdot dA_{(i)} = 0 \quad (5.11)$$

Therefore, by virtue of (5.8) and (5.11), equation (5.10) becomes:

$$K_{x(k)} = \frac{\sum_i \sum_j y_{(i,j)} \cdot \varepsilon_{f(i,j,k)} \cdot dA_{(i)}}{\sum_i \sum_j y^2_{(i,j)} \cdot dA_{(i)}} \quad (5.12)$$

Moment equilibrium around the Y – axis yields:

$$0 = \sum_i \sum_j x_{(i,j)} \cdot dF_{(i,j,k)} \Rightarrow$$

$$0 = \sum_i \sum_j x_{(i,j)} \cdot \varepsilon_{f(i,j,k)} \cdot dA_{(i)} - \varepsilon_{0(k)} \cdot \sum_i \sum_j x_{(i,j)} \cdot dA_{(i)} -$$
$$- K_{y(k)} \cdot \sum_i \sum_j x^2_{(i,j)} \cdot dA_{(i)} - K_{x(k)} \cdot \sum_i \sum_j x_{(i,j)} \cdot y_{(i,j)} \cdot dA_{(i)} \quad (5.13)$$

Similarly, by virtue of (5.8) and (5.11), equation (5.13) becomes:

$$K_{y(k)} = \frac{\sum_i \sum_j x_{(i,j)} \cdot \varepsilon_{f(i,j,k)} \cdot dA_{(i)}}{\sum_i \sum_j x^2_{(i,j)} \cdot dA_{(i)}} \quad (5.14)$$

Numerous trials with different combinations of rings and sectors has shown that the assumptions made in (5.8) and (5.11) are fully acceptable, i.e. these terms are insignificant in comparison with the other terms of the equilibrium equations; therefore, the computer program calculates $\varepsilon_{0(k)}$, $K_{x(k)}$, and $K_{y(k)}$ using equations (5.9), (5.12) and (5.14) respectively. However, in order to ensure the accuracy of the calculations, these solutions are substituted in the initial complete equilibrium equations, i.e. equations (5.7), (5.10) and (5.13). If the difference between the left and right hand side of the equations is greater than 1N or 1Nm, a warning message is included in the report.

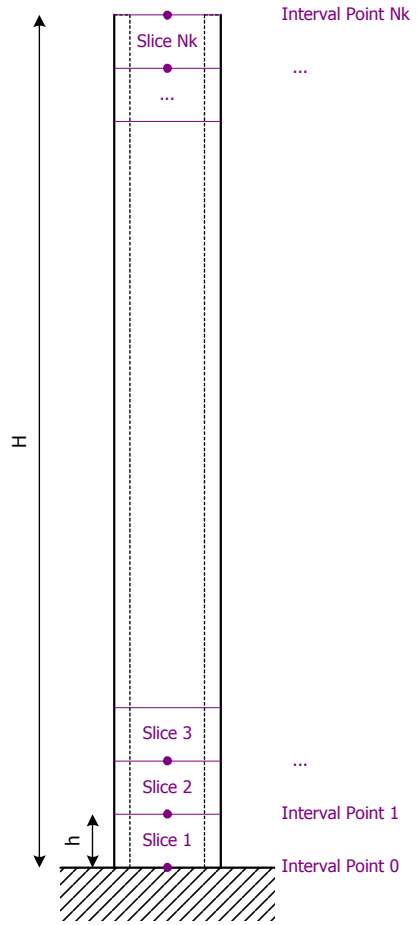
Finally, the stresses of the concrete blocks can be calculated from $\varepsilon_{0,}$ K_x and K_y:

$$\sigma_{(i,j,k)} = E_c \cdot \varepsilon_{\sigma(i,j,k)} = E_c \cdot \left(\varepsilon_{f(i,j,k)} - \left(\varepsilon_{0(k)} + K_{y(k)} \cdot x_{(i,j)} + K_{x(k)} \cdot y_{(i,j)}\right)\right) \quad (5.15)$$

5.7 Deformation calculations

5.7.1 Vertical deformation

The deformation in Z – axis is easy to calculate. We will begin the integration from the fixed base of the pier and proceed upwards; therefore:

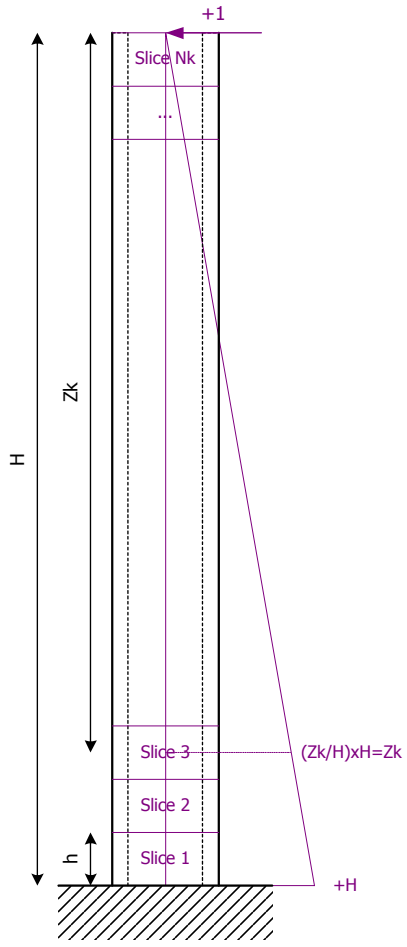


$$h = \frac{\text{Pier Height}}{\text{Number of Slices}} = \frac{H}{N_k} \quad (5.16)$$

$$\left.\begin{aligned} D_z^{(k)} &= D_z^{(k-1)} + h \cdot \varepsilon_0^{(k)} \\ D_z^{(0)} &= 0 \end{aligned}\right\} \quad (5.17)$$

5.7.2 Horizontal deflections

As a first approach, we can calculate the horizontal deflections of the tip of the pier using the principle of Virtual Work. Note that the curvature around X – axis causes *negative* deflections in the *Y* – direction and vice versa.



The distance of the centroid of the k[th] slice from the tip of the pier is:

$$z^{(k)} = \left(N_k - k + 0.5\right) \cdot h \quad (5.18)$$

Therefore:

$$m^{(k)} = \frac{z^{(k)}}{H} \cdot H = z^{(k)}$$

$$\overline{1} \cdot \Delta_y = -\oint (K_x \cdot m) dz \Rightarrow$$

$$\Delta_y = -\sum_k K_x^{(k)} \cdot z^{(k)} \cdot h \quad (5.19)$$

Similarly:

$$\Delta_x = -\sum_k K_y^{(k)} \cdot z^{(k)} \cdot h \quad (5.20)$$

Alternatively, we can calculate the deflections based on the trapezoidal rule of integration. In this way, in addition to the tip deflection, we easily obtain the *whole deflection profile* of the pier. Of course, the results for the tip deflections are the same with those provided by (5.19), (5.20). We will begin the integration from the fixed base of the pier and proceed upwards; in this way we will obtain directly the *corrected* slopes and deflections because the boundary conditions are satisfied, i.e. slope and deflection at the first interval point are zero.

The slope around the X – axis at the k^{th} interval point is:

$$h = \frac{\text{Pier Height}}{\text{Number of Slices}}$$

$$\left.\begin{aligned} S_x^{(k)} &= S_x^{(k-1)} + K_x^{(k)} \cdot h \\ S_x^{(0)} &= 0 \end{aligned}\right\} \quad (5.21)$$

The corrected deflections in the *Y – axis* caused by the above slopes are:

$$\left.\begin{aligned} D_y^{(k)} &= D_y^{(k-1)} - \frac{h}{2} \cdot \left(S_x^{(k-1)} + S_x^{(k)}\right) \\ D_y^{(0)} &= 0 \end{aligned}\right\} \quad (5.22)$$

Similarly, the slopes around Y – axis are:

$$\left.\begin{aligned} S_y^{(k)} &= S_y^{(k-1)} + K_y^{(k)} \cdot h \\ S_y^{(0)} &= 0 \end{aligned}\right\} \quad (5.23)$$

The corrected deflections in the *X – axis* caused by the above slopes are:

$$\left.\begin{aligned} D_x^{(k)} &= D_x^{(k-1)} - \frac{h}{2} \cdot \left(S_y^{(k-1)} + S_y^{(k)}\right) \\ D_x^{(0)} &= 0 \end{aligned}\right\} \quad (5.24)$$

5.8 Thermoelastic solution

The thermoelastic solution can now be obtained. The process is the following:

a) Input the data of the structure and pick the diagrams that will be used.

b) Pick the time of the day for the calculations.

c) Calculate the geometry of the structure.

d) Calculate the temperatures of each concrete block, as described in paragraph 5.5.7.

e) Calculate the free thermal strains based on equation (5.3). As there are no creep strains, the free strains are equal to the free thermal strains, equation (5.5).

f) Calculate the axial strains and curvatures of the structure, as described in section 5.6.

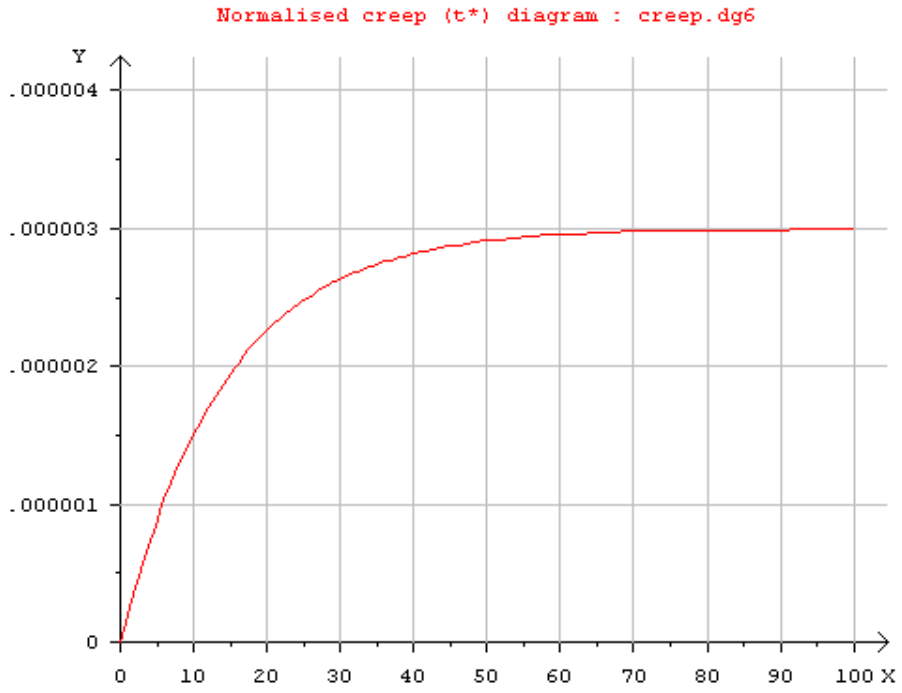
g) Calculate the deformation of the structure, as described in section 5.7.

5.9 Creep solution

In order to obtain the creep solution we need two more diagrams:

5.9.1 Diagram VI: Normalised creep diagram

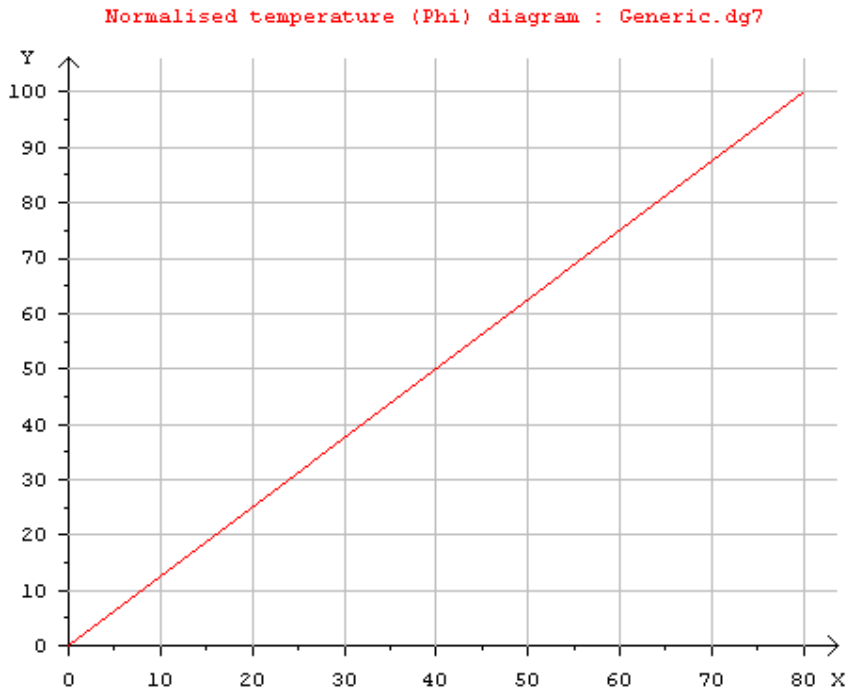
The sixth diagram provides the relation between normalised creep, i.e. pseudo – time, and real time. X values vary in the range of 0 to 100 years whereas Y values vary in the range of 0 to $4x10^{-6}$ 1/(MPa x ^{0}C), e.g.:



Therefore, for each time span in *real* time, Δt, the corresponding time span in *pseudo – time,* Δt^{*}, can be calculated using the above diagram.

5.9.2 Diagram VII: Normalised temperature diagram

The seventh diagram provides the normalised temperature function, Φ(T), as defined in the previous chapter. X values vary in the range of 0 to 80 degrees whereas Y values vary in the range of 0 to 100 degrees, e.g.:



5.9.3 Time step analysis

For the creep solution, each day is divided into a number of equal time spans. Using the equilibrium equations we can calculate the stresses for each concrete block at the time *halfway* between the start and the end of each time span. These stresses are assumed to be *constant throughout the time span*. As a result of these constant stresses, *creep strains* develop during each time span; these strains are accumulated and used in the equilibrium equations of the next time step.

The compressive creep strains that develop during the time span are:

$$\Delta\varepsilon_{c(i,j,k)} = \sigma_{\sigma(i,j,k)} \cdot \Phi\left(T_{(i,j,k)}\right) \cdot \Delta t^{*} \quad (5.25)$$

The data needed for the creep solution are the following:

a) General data of the structure.

b) Temperature – related diagrams, i.e. diagrams I to V. The computer program allows the use of different diagrams for each month of the year.

c) Normalised creep diagram and normalised temperature diagram, i.e. diagram VI and VII, corresponding to the quality of the concrete.

d) Time interval for the calculations.

e) Starting and ending date for the creep analysis.

The process is the following:

a) Calculate the geometry of the structure.

b) As a repeated process, i.e. a loop, do the following:

   i. Pick the next time span and calculate the corresponding pseudo – time span, Δt^{*}, using diagram VI.

   ii. For the instant halfway between the start and the end of the time span, calculate the temperatures of each concrete block as described in paragraph 5.5.7.

   iii. Calculate the free thermal strains based on equation (5.3).

   iv. Calculate the free strains based on equation (5.5), taking into account the creep strains that have been accumulated since the beginning of the analysis.

v. Calculate the axial strains and curvatures of the structure, as described in section 5.6.

vi. Calculate the stresses for each concrete block based on equation (5.15).

vii. Calculate the creep strains that develop during the time span using equation (5.25). Add these strains to the $\Delta\varepsilon_c$ matrix.

viii. If necessary, calculate the deformation of the structure as described in section 5.7.

6. Manual Of myCreep.exe

6.1 Capabilities

As mentioned earlier, a computer program was developed as part of this dissertation in order to obtain numerical results based on the analysis presented in the previous chapter. The computer program, called myCreep, was developed under Microsoft Visual Basic version 6 (SP3), for Microsoft Windows 95/98/2000/NT based systems.

The program is user – friendly, i.e. it offers an easy – to – use visual interface; moreover, most of the results can be plotted automatically, so that the user can check *visually* the validity of the calculations. This is extremely important in our case, as it very difficult to check the results by hand calculations.

The program is also capable of editing the diagrams directly, but we will see that it is possible to input data using a spreadsheet, e.g. Microsoft Excel. In this way, we can easily use as many points as we like.

The program can calculate the thermoelastic solution at a specified time of the day. This is important, because in this way we can check if the discretization of the structure is sufficient.

Finally, the program is able to calculate the creep solution based on the analysis presented in the previous chapter; it can also store the deflection profiles of the pier at a specified time interval.

The source code was written having both *readability* and *execution speed* in mind; therefore, the source code is not the fastest that could be written. However, the loss in speed is not significant and, because of the

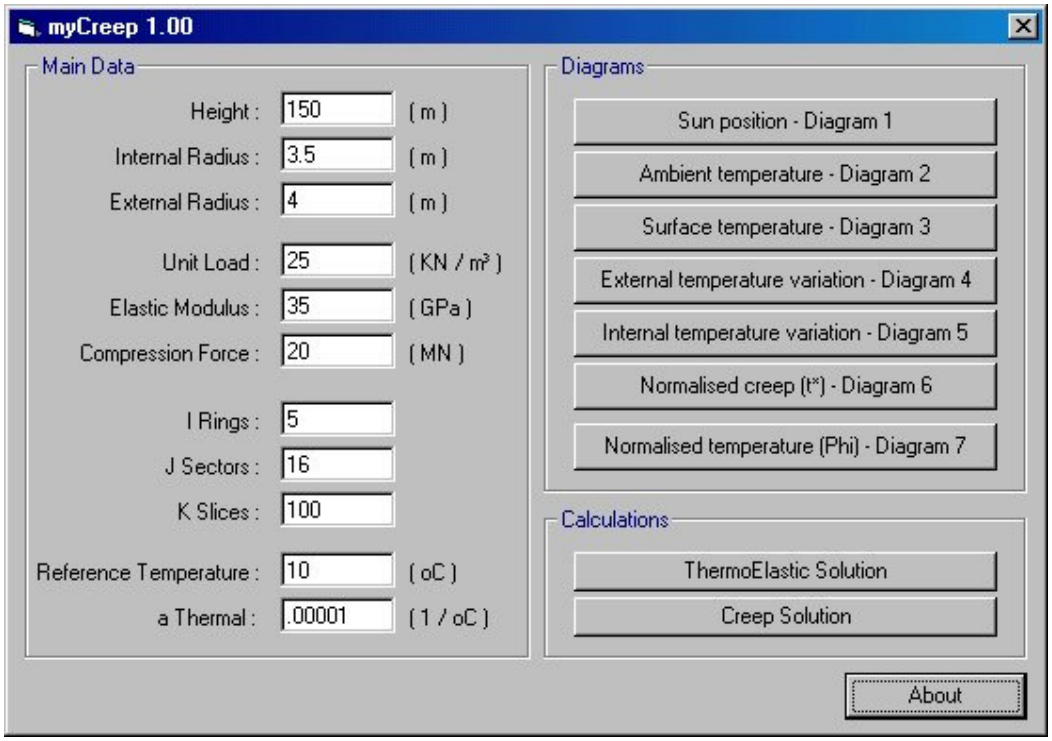
readability of the source code, *debugging*, i.e. finding and correcting errors, is easier.

For example, the interpolation routine, i.e. the routine that interpolates a value of X to obtain the Y value from a diagram, uses a bisection method. This means that if the diagram is defined by 4 points, 2 comparisons are needed; however, if the diagram is defined by *1000* points, roughly *10* comparisons are needed, since 2^{10}=1024>1000.

6.2 Data input

6.2.1 Variables

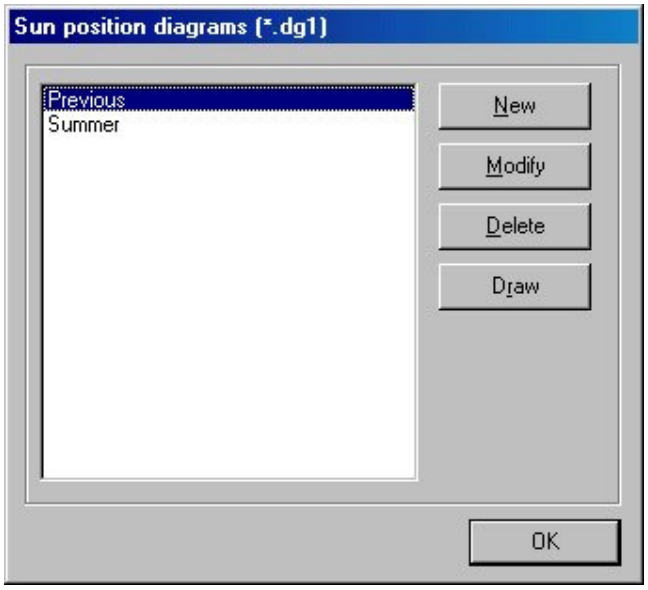
Most of the discrete variables are entered in the corresponding text boxes of the initial form:



Make sure to use the correct units.

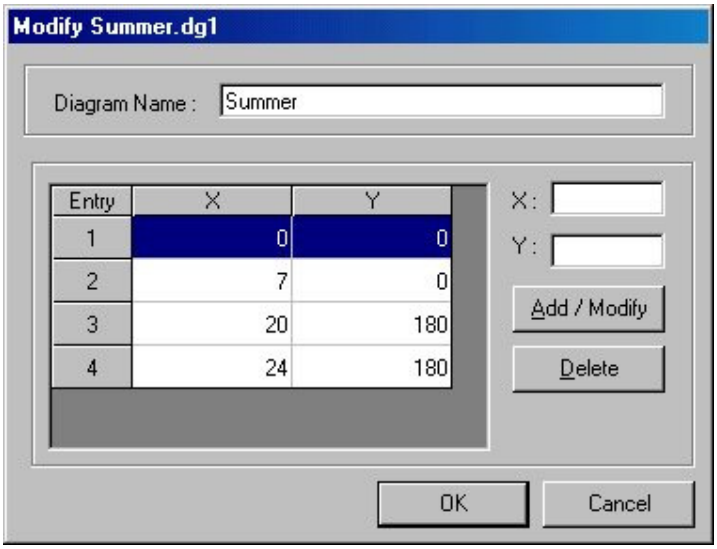
6.2.2 Diagrams

All diagrams can be edited within the program. By clicking on the appropriate button of the initial form, a list of all available diagrams will be displayed:



The options are the following:

- You can define a new diagram by clicking on the "New" button.
- You can also modify the data of a diagram, i.e. the points, by clicking on the "Modify" button. The following form will be displayed:



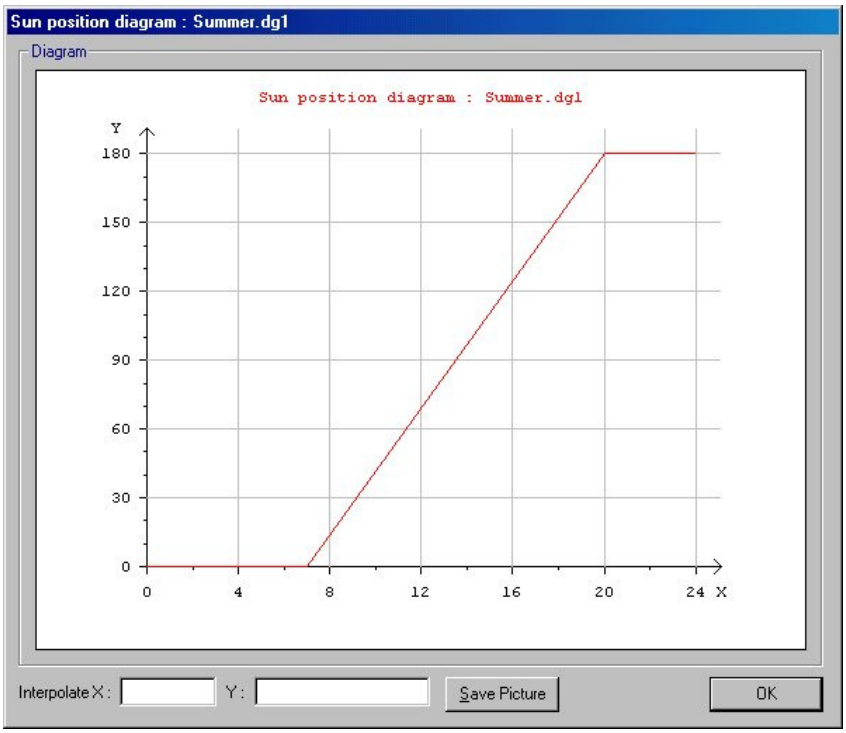
You can change the name of the diagram by writing the new name in the corresponding text box.

You can delete an entry by selecting the entry in the list and clicking on the "Delete" button.

In order to add more points or modify existing points, you must enter the X – Y values in the appropriate text boxes and click on the "Add/Modify" button. If the X value already exists, it will obtain the new Y value you specified. If the X value is not in the list, a new entry will be added. Note that the points are automatically sorted. All diagrams *must* include two entries: one for the minimum X value and one for the maximum X value. If you don't specify these points, the program will automatically add them for you.

- You can delete an existing diagram by selecting it and clicking on the "Delete" button.

- You can draw an existing diagram by selecting it and clicking on the "Draw" button. The following form will appear:



The diagram is plotted automatically; the minimum and maximum values of X and Y are fixed for each diagram.

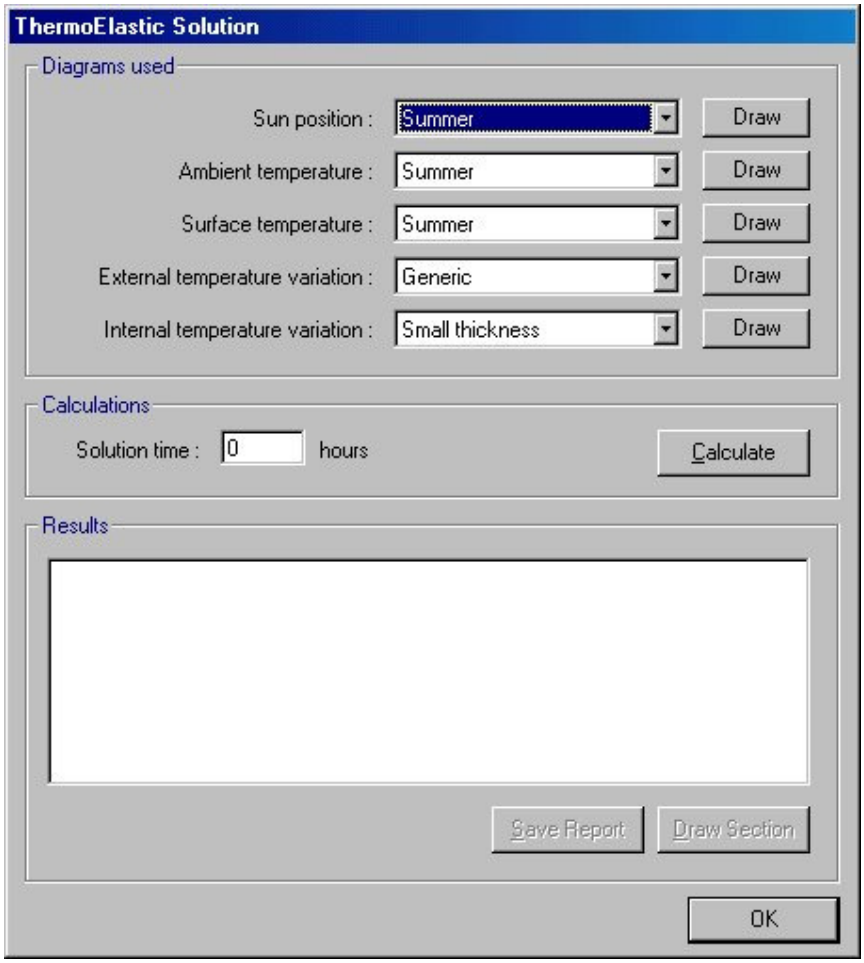
You can save the picture by clicking on the "Save Picture" button.

You can also interpolate an X value by typing in the "X" textbox; the result is automatically displayed in the "Y" text box. The same interpolation routines that are used in the main calculations are used here as well.

As mentioned earlier, the diagrams are stored in the application's path as individual files with the extension ".dg1" to ".dg7", depending on the type of the diagram. These files are simple tab – delimited ASCII text files; therefore you can easily create them using a text editor like Notepad. Moreover, you can calculate points using a spreadsheet, e.g. Microsoft Excel, and save the file as a tab – delimited text file. Make sure that the file is clean from redundant data and "invisible" tabs; the program will erase automatically all entries that are not of the form "X value – TAB – Y value – CR".

6.3 Thermoelastic solution

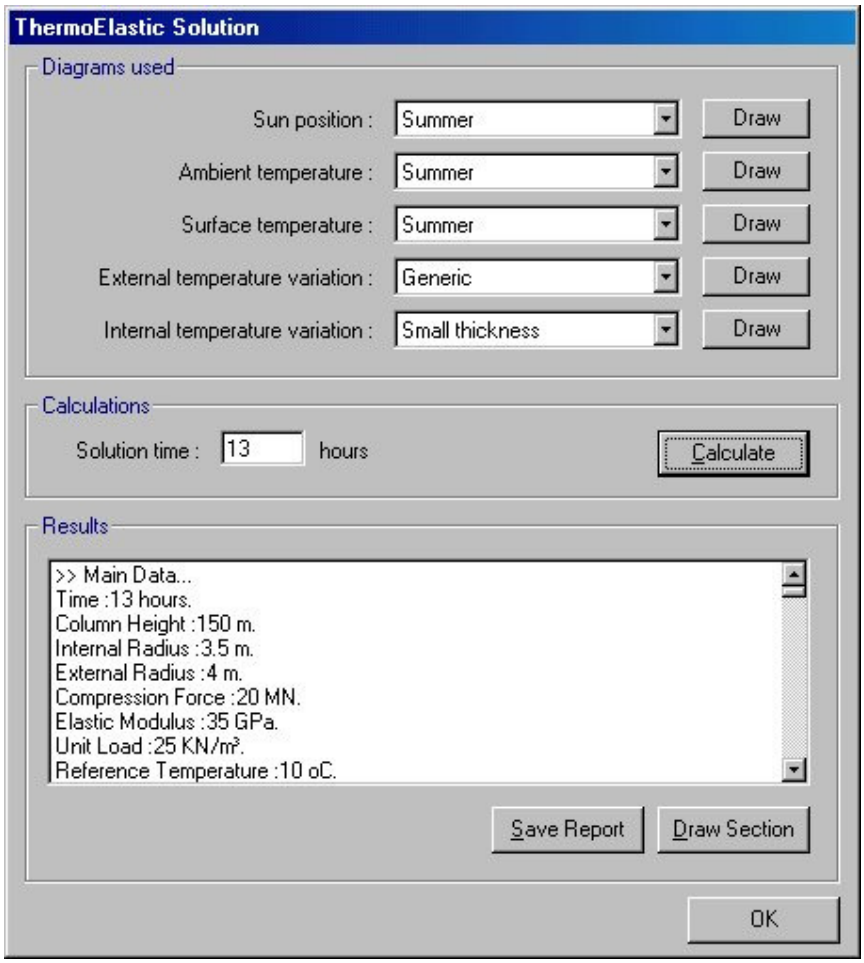
In order to obtain the thermoelastic solution you must click on the corresponding button of the initial form. The following form will appear:



You must pick the temperature – related diagrams by clicking on the corresponding drop – down lists. You can draw a diagram by selecting it and clicking on the corresponding "Draw" button.

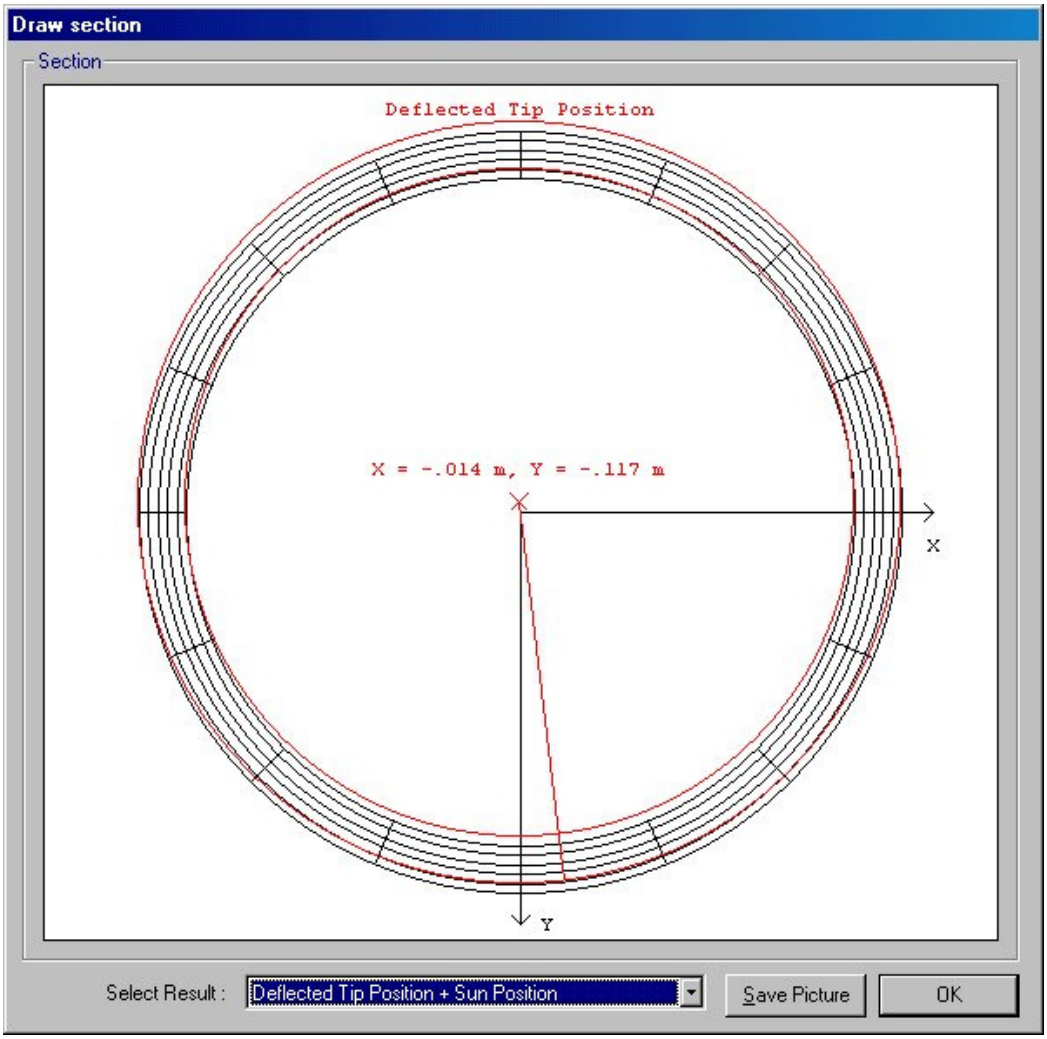
You must also select the time of the day for the calculations, by filling the corresponding text box.

Finally, click on the "Calculate" button; the results list will be filled and the buttons "Save Report" and "Draw Section" will be enabled:



You can save the contents of this list by clicking on the "Save Report" button.

You can also check the results *visually* by clicking on the "Draw Section" button. The following form will appear:

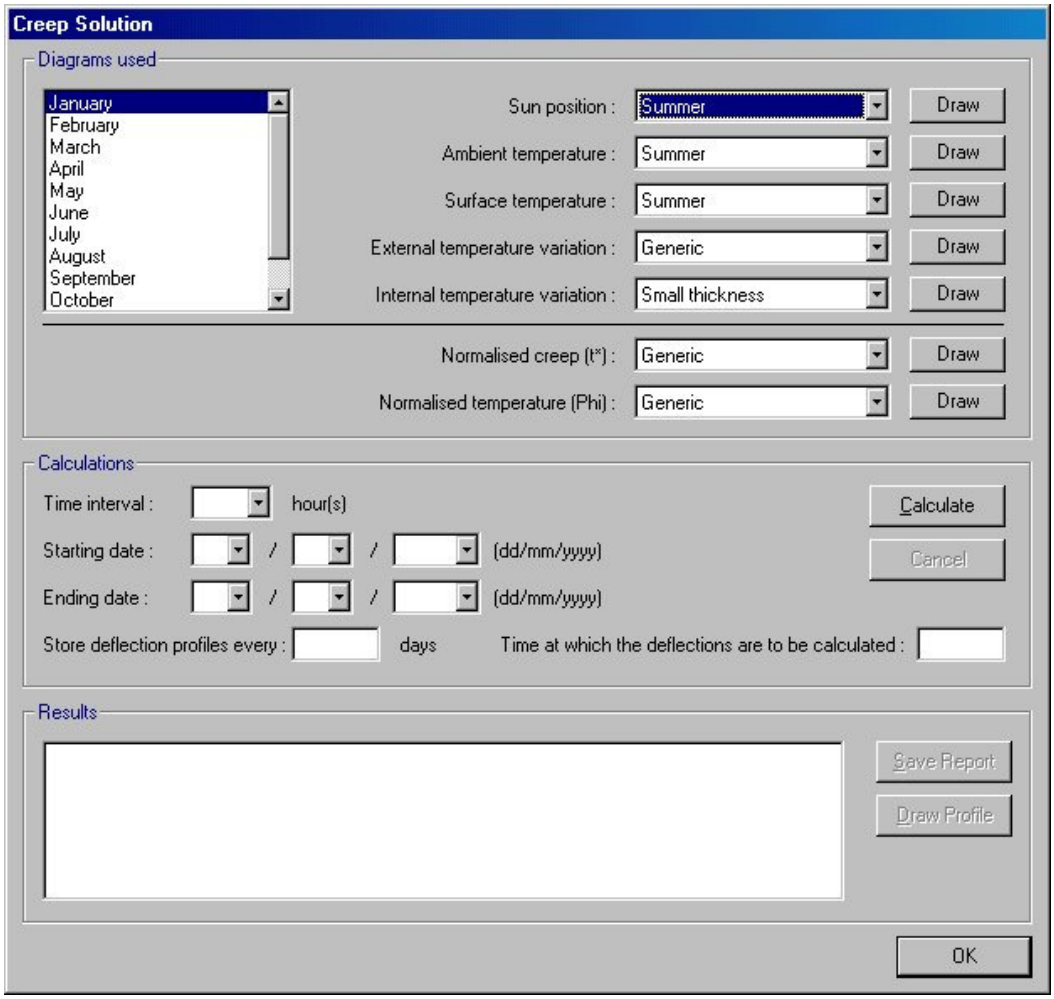


You can select the result that you want to be displayed by clicking on the corresponding drop – down list. The results include block temperatures, the position of the sun, deflected tip position etc. For example, the above picture illustrates the deflected position of the tip of the pier and the position of the sun at the same time. It is important to notice that the line that shows the position of the sun passes through the origin. This means that the pier deflects *directly against the sun*, which is logical; therefore, the discretization of the structure, i.e. the number of rings and sectors, seems to be sufficient.

You can save the picture by clicking on the "Save Picture" button.

6.4 Creep solution

In order to obtain the creep solution you must click on the corresponding button of the initial form. The following form will appear:

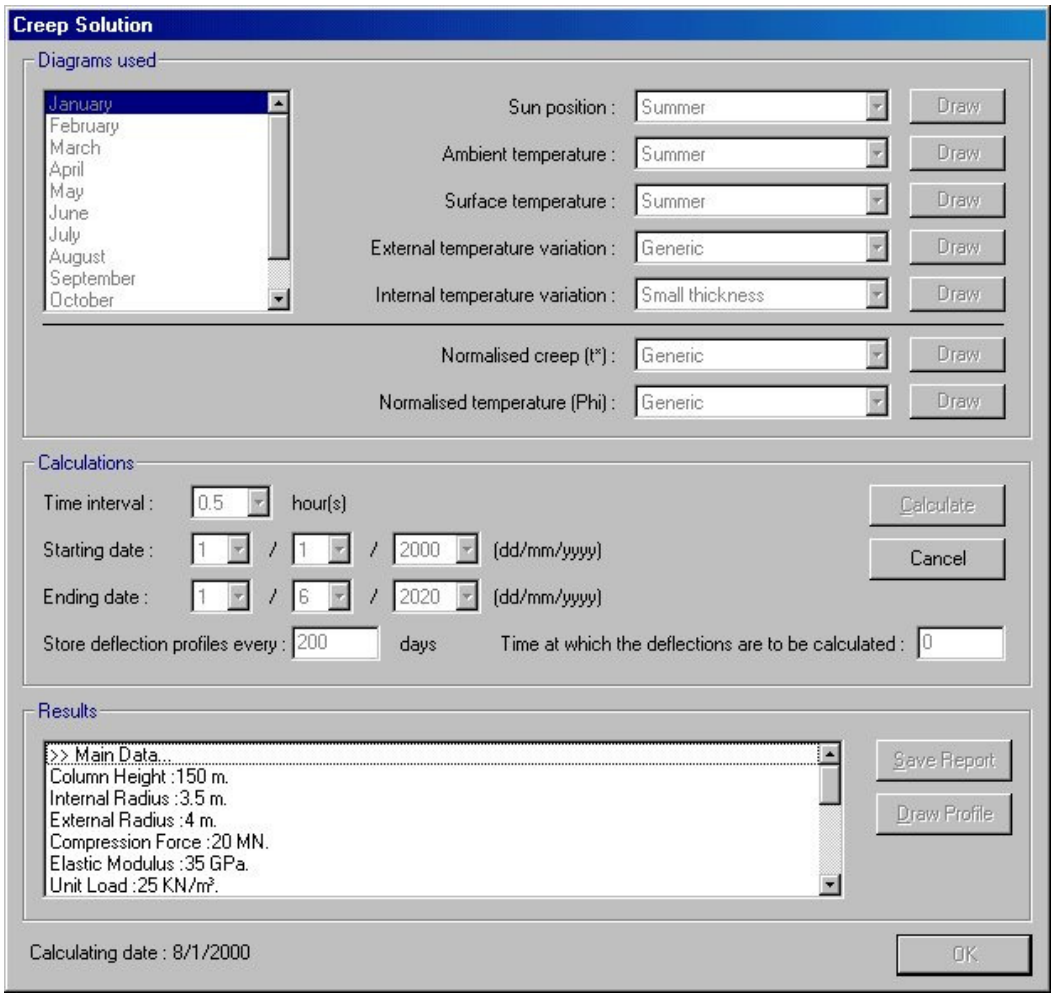


You must pick the temperature – related diagrams for each month of the year. This is accomplished by *first* selecting the month and *then* clicking on the corresponding drop – down lists. You must also pick the normalised creep and normalised temperature diagrams. You can draw any diagram by selecting it and clicking on the corresponding "Draw" button.

You must also select the time interval, the starting and ending date for the calculations and the time of the day for the deflection profiles to be

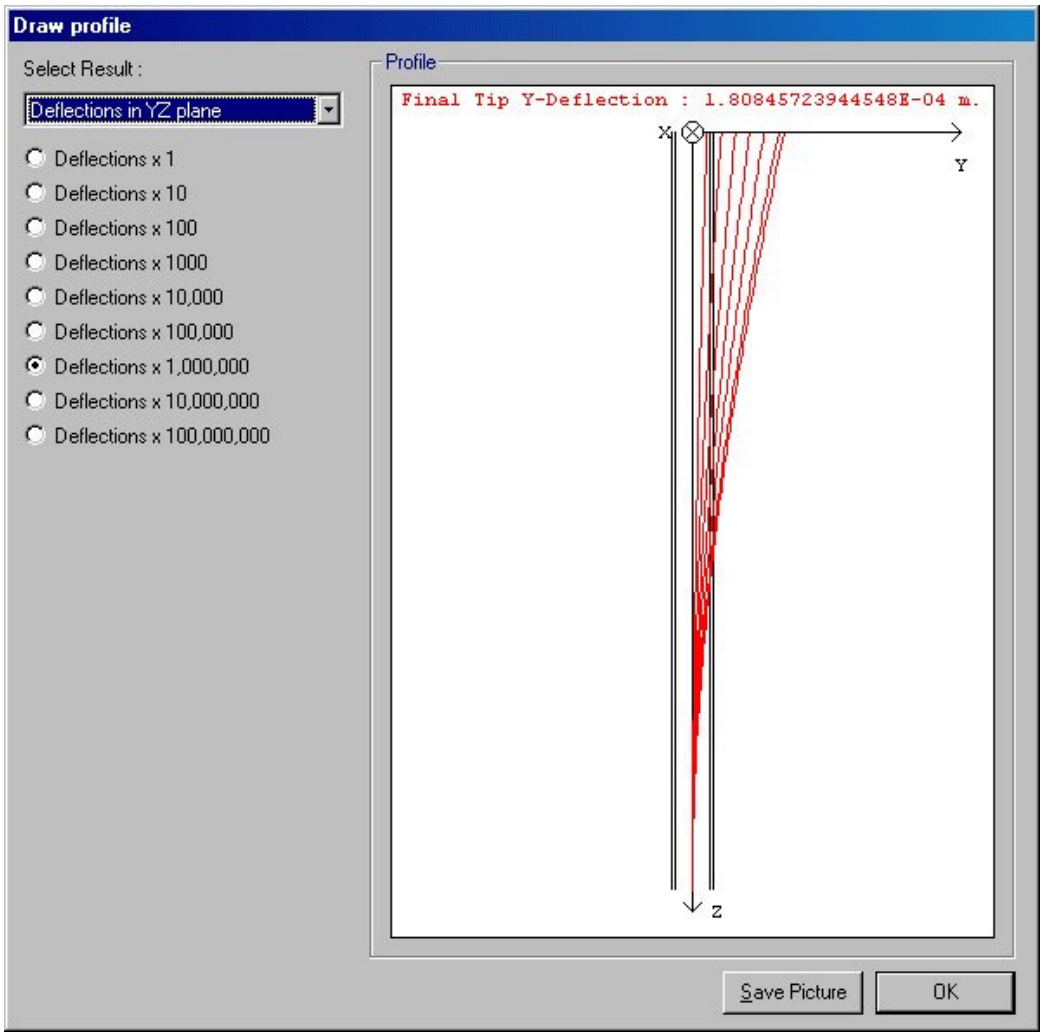
calculated. It would be better to choose a time when the ambient temperature and the surface temperature are the same, e.g. at night; in this way one can monitor the effects of creep *alone* on the horizontal deflections. Optionally, you can pick the time interval for the storage of the deflection profiles; the program will automatically store the deflection profile at the end of the calculations.

Finally, click on the "Calculate" button; the program will begin the calculations and the results list will be gradually filled. A label at the bottom left corner of the form shows the current date of the calculations:



You can abort the calculations at any time by clicking on the "Cancel" button. After the completion of the calculations, the "Save Report" and "Draw Profile" buttons will be enabled. You can save the contents of the results list

by clicking on the "Save Report" button. You can also check the results *visually* by clicking on the "Draw Profile" button. The following form will appear:



You can select to view the deflection profiles in YZ or ZX plane by clicking on the corresponding drop – down list. You can also use the magnification factors provided in order to distinguish the profiles.

Finally, you can save the picture by clicking on the "Save Picture" button.

7. Numerical Example

7.1 Brief

We will now obtain the creep solution of a slender cylindrical concrete pier for a 50 years' period. The data used don't correspond to a real structure; the diagrams are also imaginary. However, the example shows clearly the advantages of this numerical step – by – step creep analysis.

The deflection profiles were stored at midnight, so that the ambient temperature and the surface temperature are the same. In this way, the effects of creep on the horizontal deflections are clear.

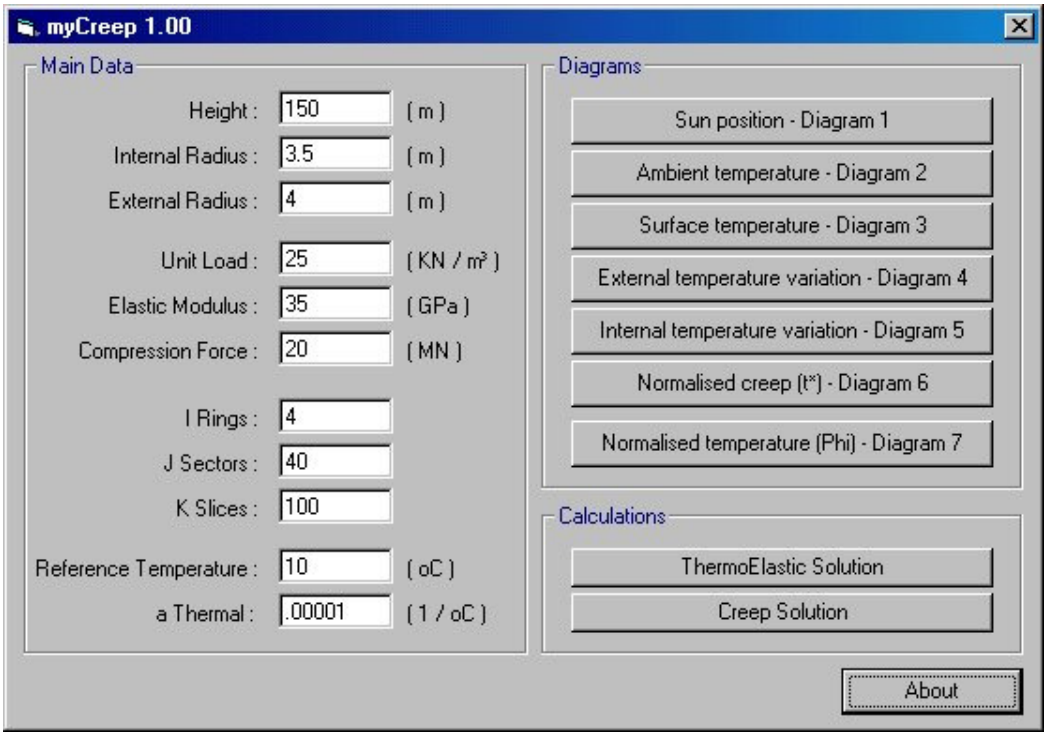
7.2 Data

7.2.1 Variables

The data used in this example are:

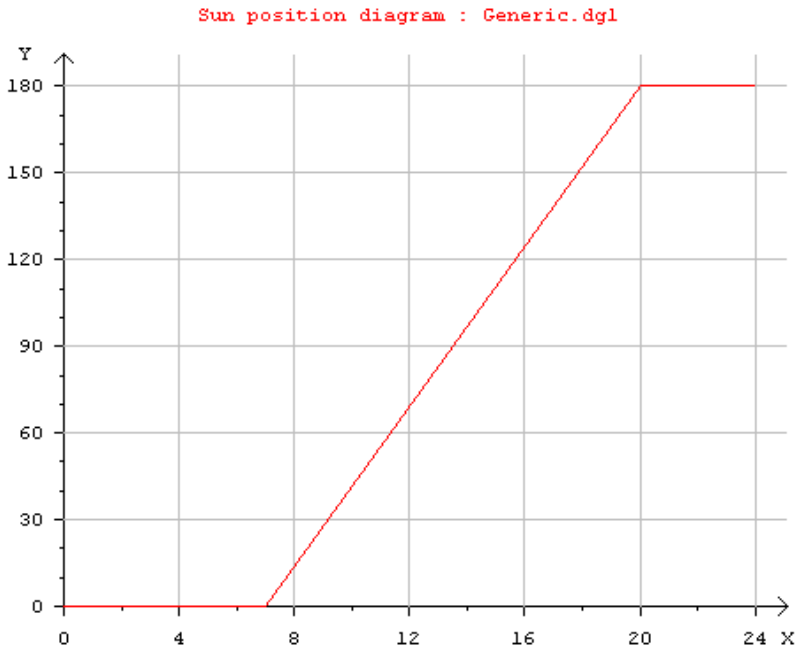
- Height: 150 m
- Internal radius: 3.5 m
- External radius: 4 m
- Compressive force: 20 MN
- Unit load: 25 KN/m^3
- Elastic modulus: 35 GPa

- Reference temperature: 10 ^{0}C
- Coefficient of thermal expansion: 10 x 10^{-6} 1/^{0}C
- The structure was divided into 4 rings, 40 sectors and 100 slices.

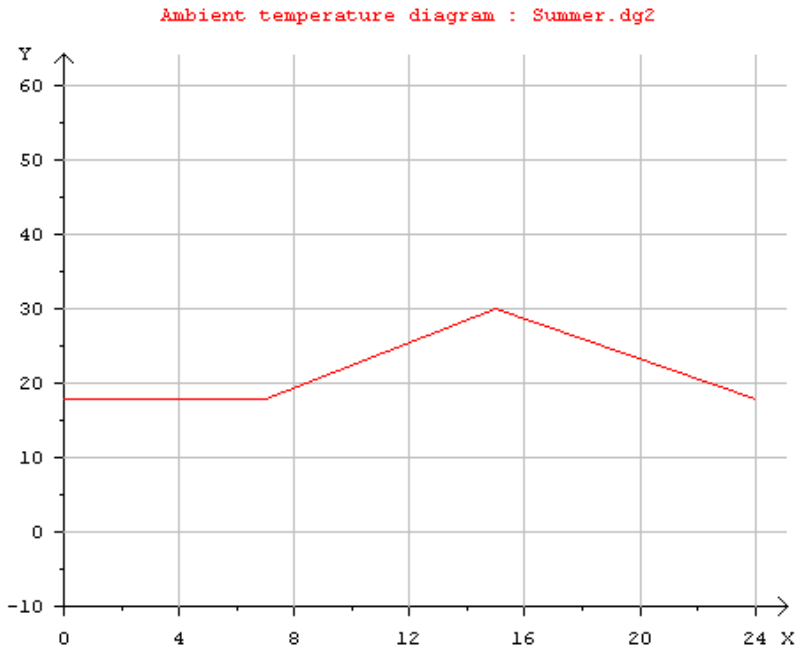


7.2.2 Diagrams

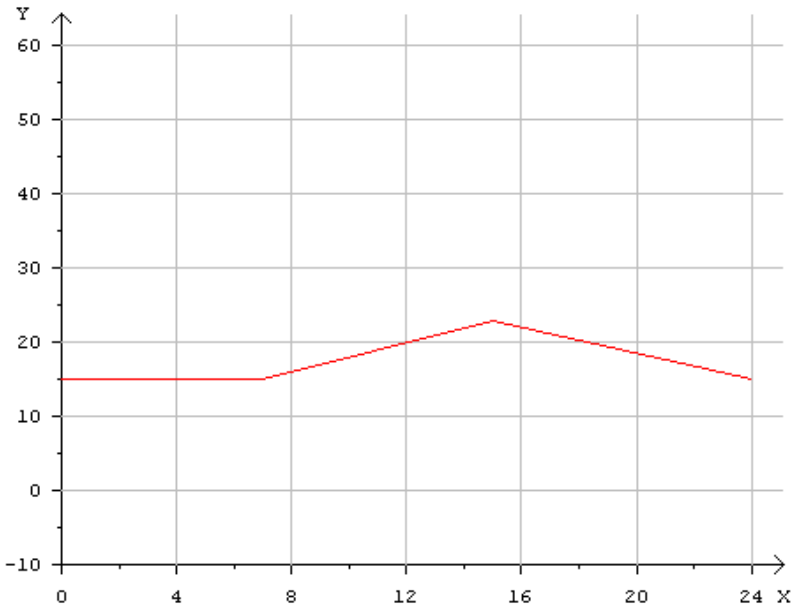
- The same sun position diagram was used for all months:



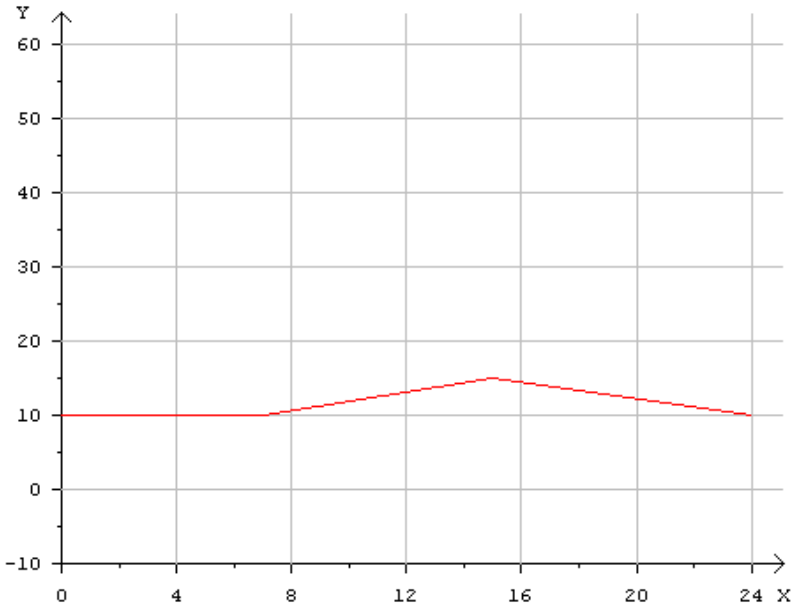
- Four different ambient temperature diagrams were used, one for each season:

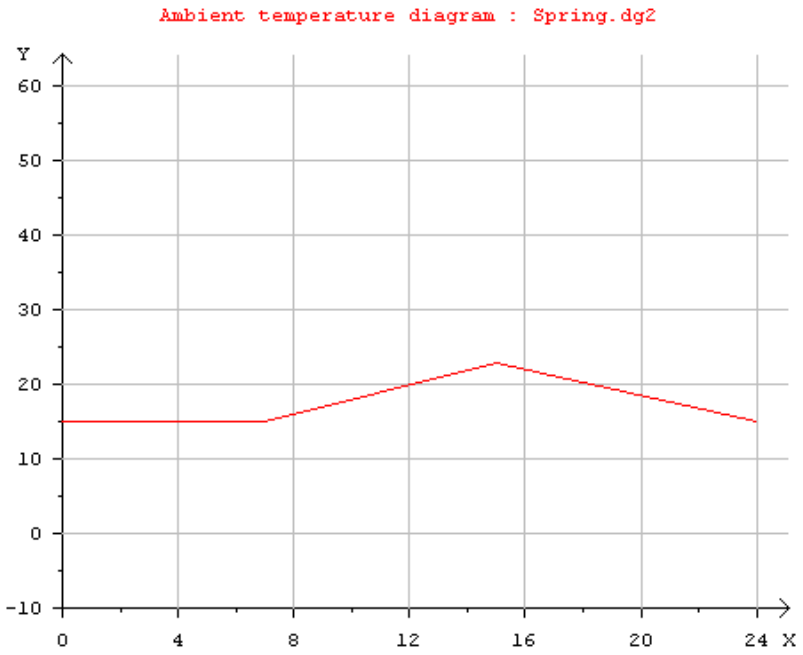


Ambient temperature diagram : Autumn.dg2



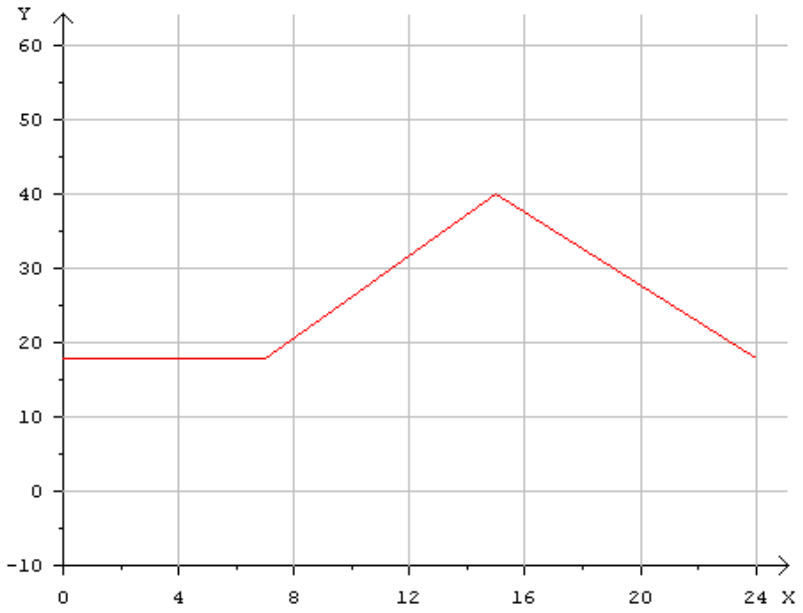
Ambient temperature diagram : Winter.dg2



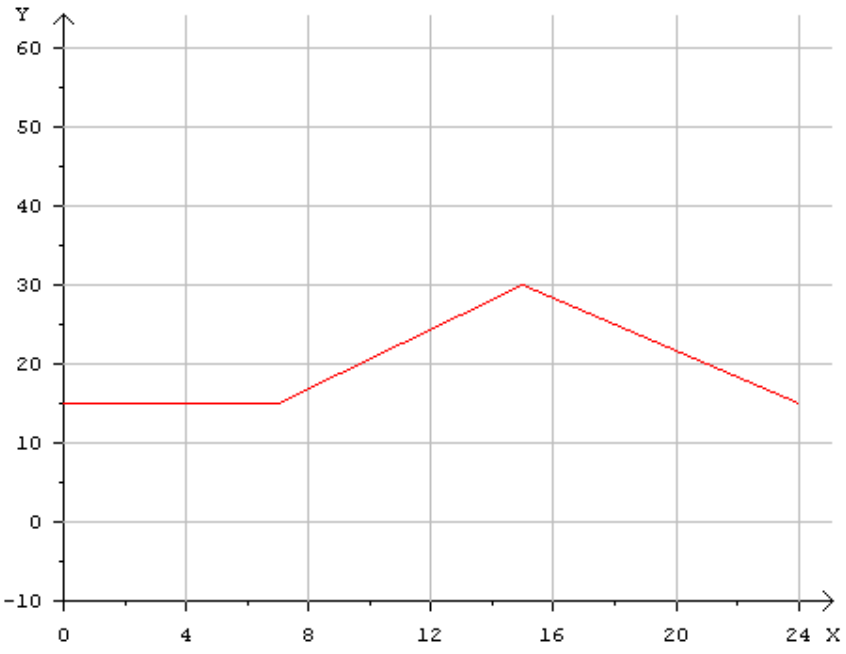


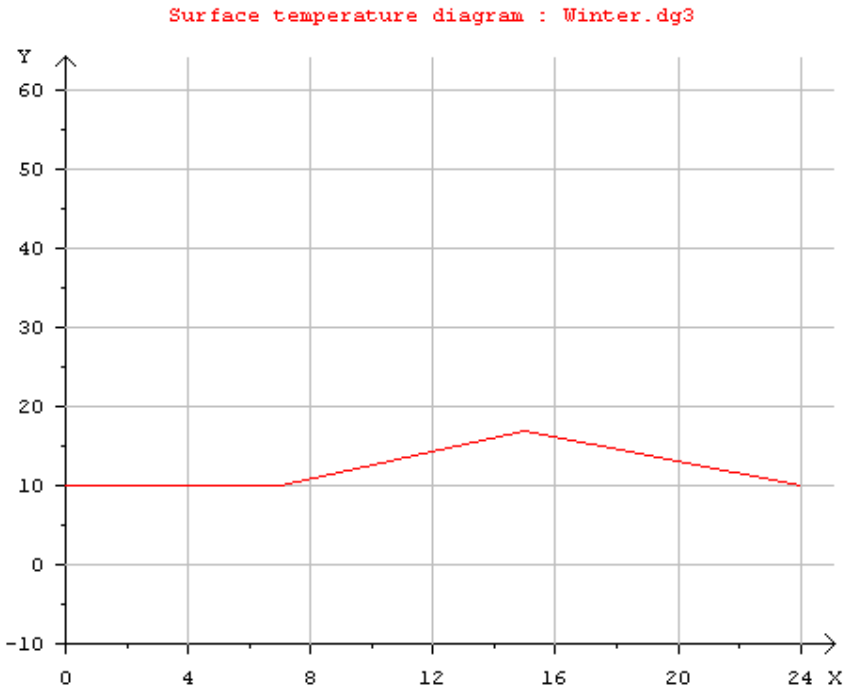
- Similarly, four different surface temperature diagrams were used, one for each season. Note that the surface temperature must be greater than or equal to the corresponding ambient temperature at all times:

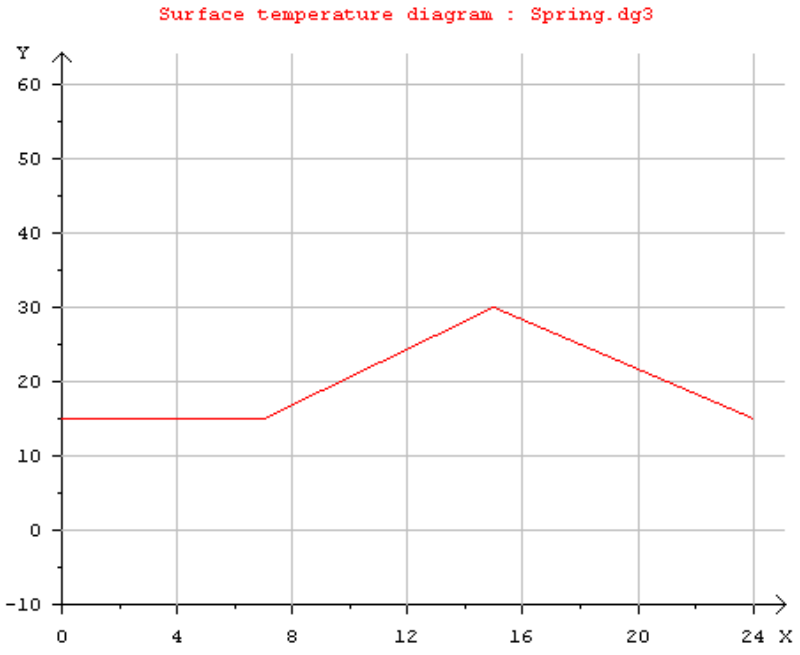
Surface temperature diagram : Summer.dg3



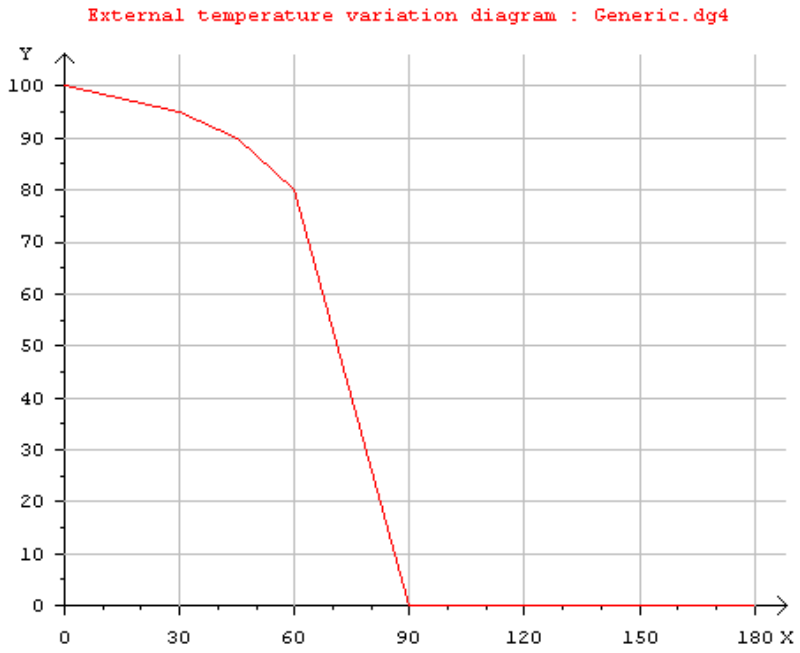
Surface temperature diagram : Autumn.dg3



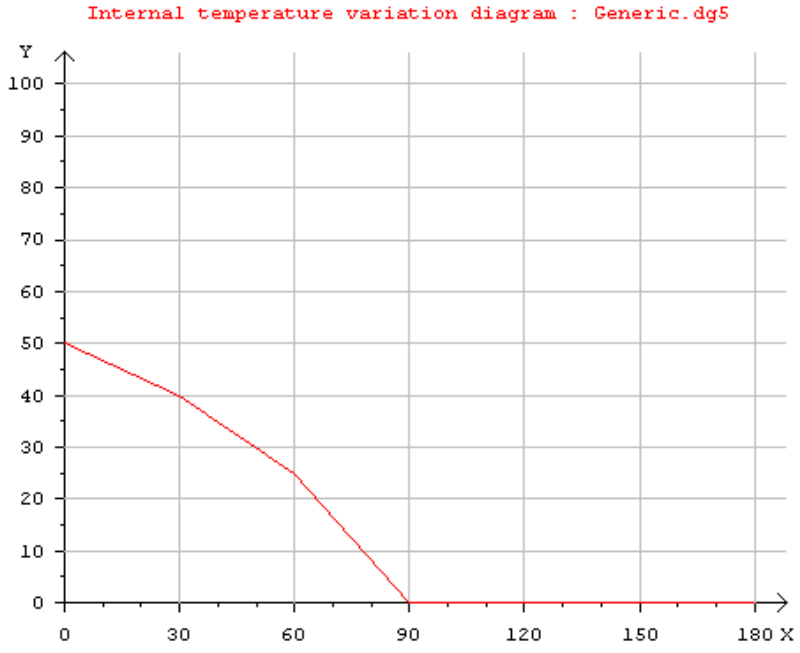




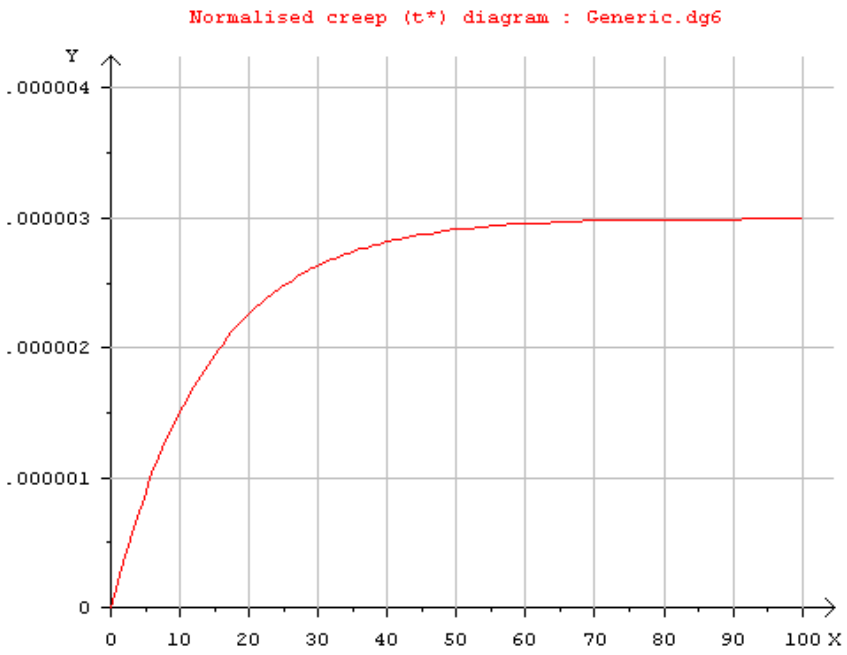
- The following external variation diagram was used for all months:



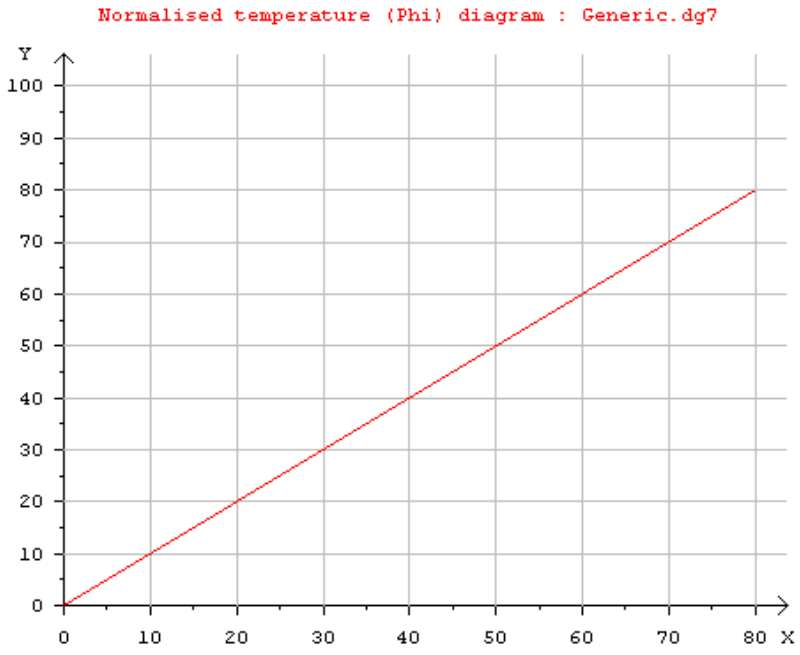
- The following internal variation diagram was used for all months:



- The following normalised creep diagram was used:



- Finally, the following normalised temperature diagram was used:

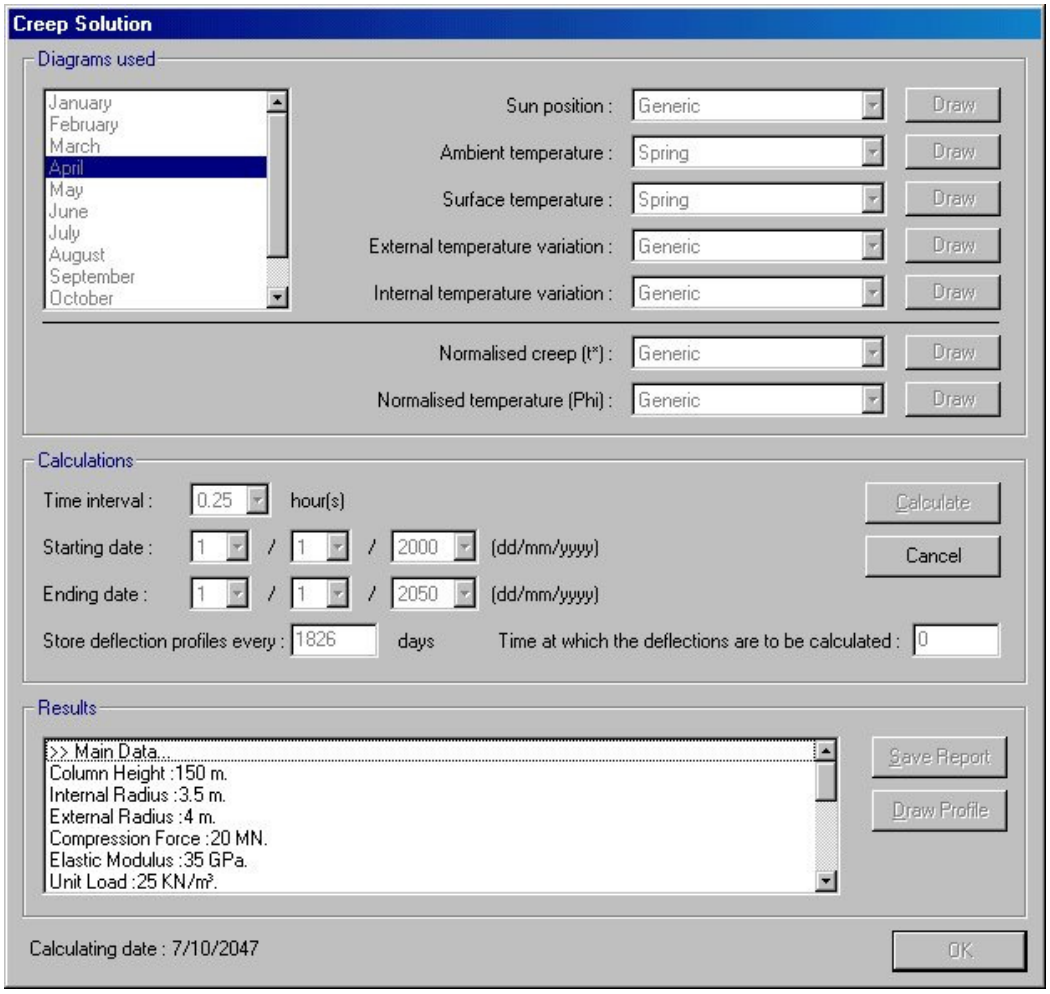


7.3 Calculations

The following data were also used for the creep analysis:

- Time interval: a quarter of an hour (0.25 hours)
- Starting date: 1/1/2000
- Ending date: 1/1/2050
- Deflection profiles were stored every 5 years.
- Deflection profiles were stored at midnight (0 hours).

The analysis lasted almost 12 hours using a Pentium III 500 MHz computer system:



7.4 Results

The results from this analysis are the following:

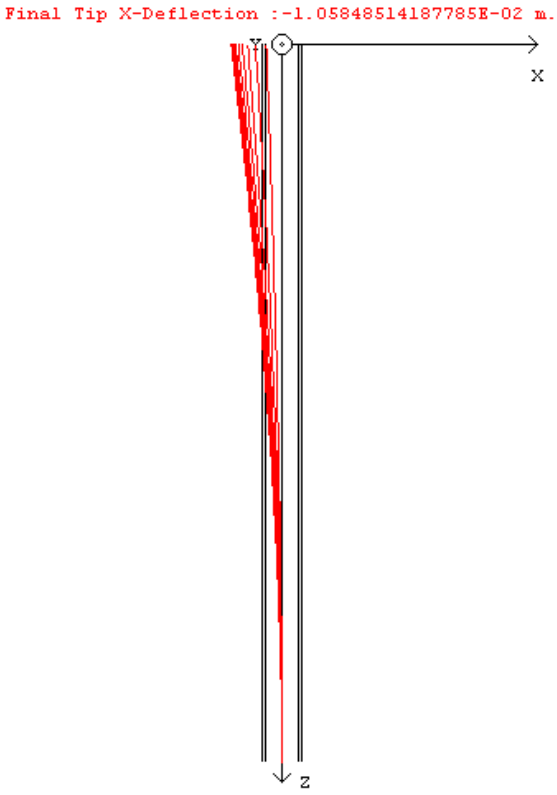
```
>> Main Data...

Column Height :150 m.

Internal Radius :3.5 m.

External Radius :4 m.

Compression Force :20 MN.

Elastic Modulus :35 GPa.

Unit Load :25 KN/m³.

Reference Temperature :10 oC.
```

```
Coefficient of thermal expansion :.00001 1/oC.
Number Of Rings :4
Number Of Sectors :40
Number Of Slices :100
>> Calculating Geometry...
>> Preliminary Calculations...
>> Initialising...
Calculating from :1/1/2000 to 1/1/2050 (dd/mm/yy)
Time Interval :.25 hour(s).
Storing deflection profiles every :1826 days.
Time of day (for the profiles) :0 hour(s).
>> Tip Deflection Calculations...
Date : 30/12/2004
X - Deflection :-3.26572394325431E-03 m.
Y - Deflection :7.18211694062574E-03 m.
Z - Deflection :2.38054630059606E-02 m.
Date : 30/12/2009
X - Deflection :-5.51506794449949E-03 m.
Y - Deflection :1.21329523752215E-02 m.
Z - Deflection :.029762326979823 m.
Date : 30/12/2014
X - Deflection :-7.11014565368568E-03 m.
Y - Deflection :1.56450353421036E-02 m.
Z - Deflection :3.39935780883872E-02 m.
Date : 30/12/2019
X - Deflection :-8.2419271755086E-03 m.
Y - Deflection :1.81363405483911E-02 m.
Z - Deflection :.036986583984502 m.
Date : 29/12/2024
X - Deflection :-9.04123888118818E-03 m.
Y - Deflection :1.98970117945515E-02 m.
Z - Deflection :3.91071335835956E-02 m.
```

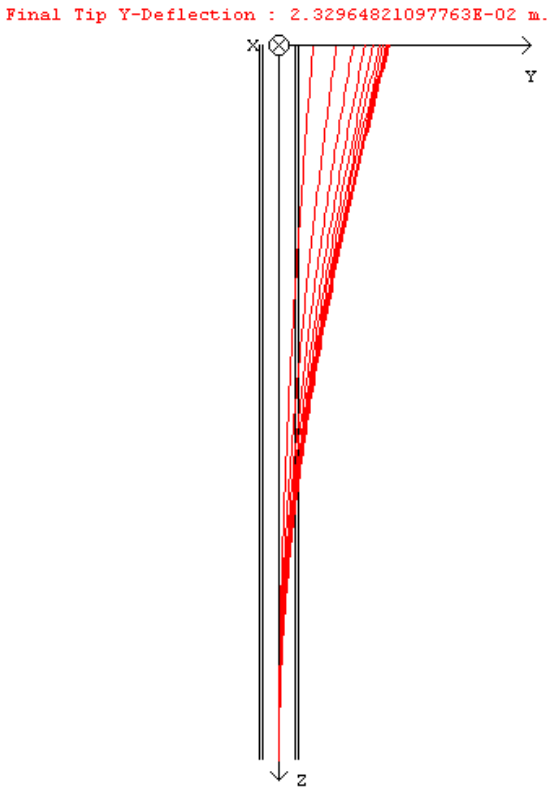
```
Date : 29/12/2029
X - Deflection :-9.59035519493462E-03 m.
Y - Deflection :2.11056119903369E-02 m.
Z - Deflection :4.05581617583533E-02 m.
Date : 29/12/2034
X - Deflection :-9.98624456626175E-03 m.
Y - Deflection :2.19773413284071E-02 m.
Z - Deflection :4.16135148350967E-02 m.
Date : 29/12/2039
X - Deflection :-1.02690352683807E-02 m.
Y - Deflection :2.26006805865748E-02 m.
Z - Deflection :4.23747177780371E-02 m.
Date : 28/12/2044
X - Deflection :-1.04396714975139E-02 m.
Y - Deflection :2.29764743938087E-02 m.
Z - Deflection :4.28429006616023E-02 m.
Date : 28/12/2049
X - Deflection :-1.05848514187785E-02 m.
Y - Deflection :2.32964821097763E-02 m.
Z - Deflection :4.32283147013877E-02 m.
Date : 1/1/2050
X - Deflection :-1.05848514187785E-02 m.
Y - Deflection :2.32964821097763E-02 m.
Z - Deflection :4.32283147013877E-02 m.
```

The deflection profiles, stored every 5 years, are the following:

- In the ZX plane, (X deflections magnified by a factor of 10000):



- In the YZ plane (Y deflections magnified by a factor of 10000):



7.5 Comments

Some comments can now be made based on the results:

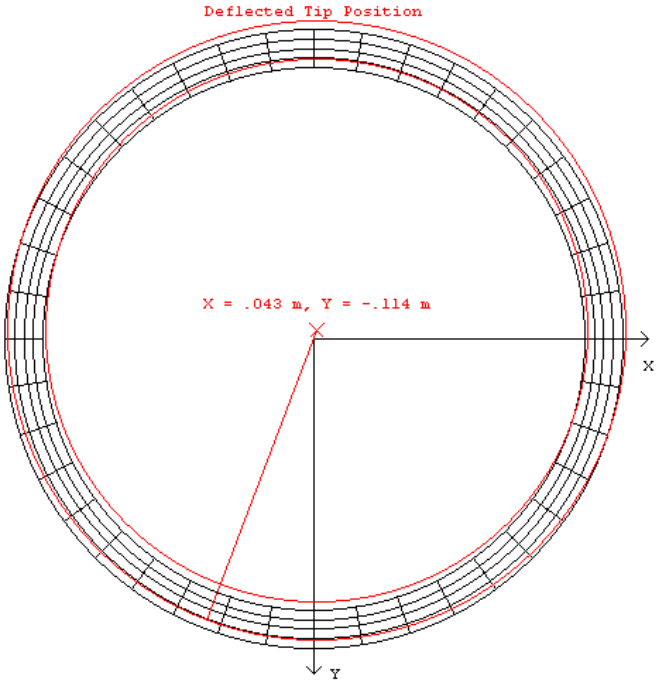
i) As expected, the deflection profiles are sparser at the beginning of the analysis when the creep rate is greater. However, as time passes and creep rate reduces, the deflection profiles become increasingly dense.

ii) As expected, the tip of the pier deflects towards a south – southwest direction. This is because of the diagrams that were used; all ambient temperature and surface temperature diagrams had an almost symmetrical arrangement around 15:00 hours, i.e. in the range of 07:00 - 15:00 – 24:00 hours. According to the sun position diagram, i.e. diagram I, the position of the sun at 15:00 hours is at 110.769^0, where 0^0 is East and 90^0 is South. Therefore, the result is logical. Note that the pier does not deflect *exactly* at 110.769^0. This is so for two reasons: first, the temperature diagrams have an *almost* symmetrical variation around 15:00 hours; second, the sun position diagram is *not symmetrical in the same range* as the temperature diagrams. If we had used symmetrical temperature diagrams in the same range as the sun position diagram, i.e. 07:00 – 13:30 – 20:00 hours, the tip of the pier would deflect *exactly* towards 90^0, since at 13:30 hours the sun position is at 90^0.

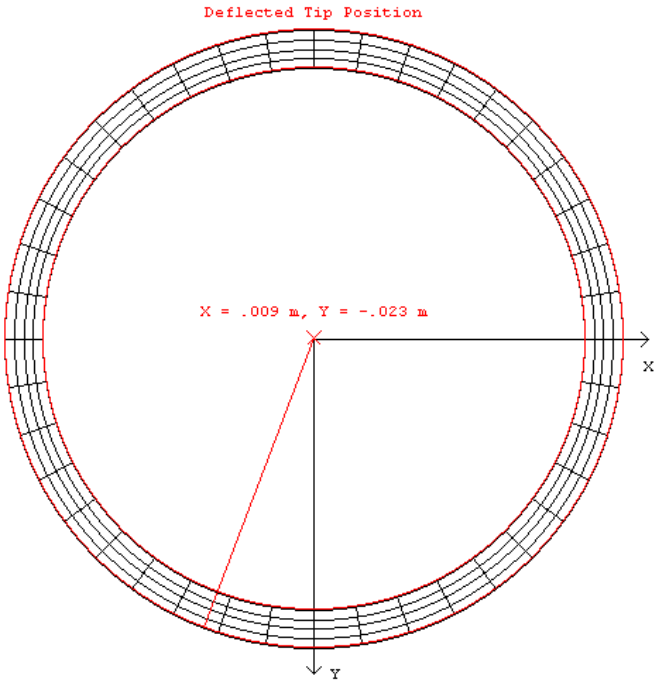
iii) It is very important to note that, while creep deflections are generally towards *South,* thermoelastic deflections are generally towards *North*. This is so because the southern part of the pier is generally hotter; therefore it creeps more, causing the pier to deflect in a Southern direction. However, the thermoelastic

deflection of the pier is *against* the sun, i.e. in a Northern direction.

iv) The X and Y – deflection of the tip of the pier because of creep alone were found to be ~-1.06 cm and ~2.33 cm, respectively, after 50 years. Therefore, the overall deflection is ~2.56 cm. We can compare this value with the thermoelastic deflection in the summer, at the hotter time of the day i.e. at 15:00 hours:



In this case, the overall deflection is ~12.18 cm. Therefore, in this case, the creep deflection is ~21% of the thermoelastic deflection. However, for the hotter time of the day in winter, the creep deflection is *greater* than the thermoelastic deflection:



Of course, these deflections are not based on experimental data and they are not in the same direction; therefore, they cannot be directly compared. The purpose of the comparison was to obtain the relative magnitude of these values.

v) As far as the horizontal creep deflections are concerned, the critical factor is not the *magnitude* of the temperatures but the *difference* between the ambient temperature and the surface temperature. This means that the critical factor is *sunshine*, which heats concrete above the ambient temperature. The corresponding diagrams used in this example were not conservative; bearing in mind that, for example, some areas in the Mediterranean have as many as 320 days of sunshine per year, it is obvious that in these cases creep deflections may be significantly greater.

8. Closing Remarks

8.1 Conclusion

Based on the previous analysis it is obvious that non – uniform temperatures caused by solar heating can cause *significant* thermoelastic and long – term creep deflections in a slender cylindrical concrete pier. As far as the horizontal creep deflections are concerned, the critical factor is *sunshine*, which heats concrete above the ambient temperature. For many areas where there are many days with sunshine per year, the deflections caused by creep maybe a critical consideration in design.

Numerical step – by – step creep analysis provides an excellent tool for dealing with these problems. Combined with a suitable creep law and realistic temperature data, this analysis can provide very good results; the main uncertainty lies in creep data of the specific concrete of the structure. However, a conservative approach can provide excellent engineering bounds to the given problem for all times.

8.2 Topics for further discussion

As a first improvement of the computer program, we could include the option to calculate the deformation and the stressing based on Burgars law. In this way we would be able to compare the results obtained from the two creep laws.

Apart from creep, *shrinkage* of concrete can also be taken into account. Shrinkage depends on temperature but also on the position with respect to the surface of concrete; concrete shrinks more if it is closer to the

surface. By adopting diagrams similar to those used for the calculation of temperatures, we could relate shrinkage with the position of a concrete block.

Finally, many concrete piers are not cylindrical but *rectangular* in cross section. One can develop a computer program to deal with this kind of piers, adopting a similar approach.

8.3 Bibliography

8.3.1 Creep

i) *"A temperature – creep theory for prestressed concrete continuum and beam structures"*, paper by G. L. England, Y. F. Cheng and K. R. F. Andrews, Journal of Thermal Stresses 7:361 – 381.

ii) *"Micro – computer aided design and construction of segmental prestressed concrete cantilever bridges"*, key – note lecture delivered to the 2nd international conference on Computer Applications In Concrete, Singapore, March 1988.

iii) *"Time – dependent stresses in creep – elastic materials: a general method of calculation"*, paper by G. L. England, conference on "Recent advances in stress analysis: new concepts and techniques and their practical application", 26th – 29th March 1968.

iv) *"Volume Changes and Creep of Concrete"*, by R.F. Feldman, Canadian Building Digest, CBD – 119.

v) *"Time and temperature dependent behaviour of concrete structures"*, notes taken from lectures by Pr. G. L. England

(1999-2000 MSc/DIC in Concrete Structures, Imperial College, London).

vi) *"Concrete materials"*, notes taken from lectures by Dr. Newman (1999-2000 MSc/DIC in Concrete Structures, Imperial College, London).

8.3.2 Programming

i) *"Visual Basic 5.0 Programmer's Guide to the Win32 API"*, by Dan Appleman, Ziff – Davis Press.

ii) *"Active Visual Basic 5.0"*, by G. Eddon, Mixed Media, Microsoft Press International, 1997.

iii) *"Print Programming in Windows"*, by Jeff Potts, R & D Books.

9. Appendix: Source Code

The following source code is *part* of the complete program (1766 out of a total of 3781 lines). However, almost all calculation routines are included.

The source code presented here is for reference only and it is provided "as is". Please note that the author is not responsible and assumes no liability whatsoever for any results, or any use made of the results obtained from the program.

modDiagrams.bas:

```
Option Explicit

Public Type PointAPI
    X As Long
    Y As Long
End Type

Public Type PointInfo
    X As Double
    Y As Double
End Type

Public Type DiagramInfo
    Entries As Long
    Points() As PointInfo
End Type

Public Const NumOfDiagrams = 7
Public minX(1 To NumOfDiagrams) As Double
Public maxX(1 To NumOfDiagrams) As Double
Public minY(1 To NumOfDiagrams) As Double
Public maxY(1 To NumOfDiagrams) As Double

Public Enum DiagramResults
    ERROR_INVALIDPOINT = -2
    ERROR_FILENOTFOUND = -1
    ERROR_CRITICAL = 0
    OK = 1
    OK_ERRORFIXED = 2
End Enum

Public EditDiagram As DiagramInfo
Public EditDiagramName As String
Public EditDiagramMode As Long

Public XmajTick(1 To NumOfDiagrams) As Single
Public XminTick(1 To NumOfDiagrams) As Single
Public YmajTick(1 To NumOfDiagrams) As Single
Public YminTick(1 To NumOfDiagrams) As Single
Public DiagramTitle(1 To NumOfDiagrams) As String

Public Diagram(1 To NumOfDiagrams) As DiagramInfo
```

```
Public crMonthName(1 To 12) As String
Public crDaysOfMonth(1 To 12) As Long

Public Sub InitialiseValues()

    'Sun position
    DiagramTitle(1) = "Sun position"
    minX(1) = 0
    maxX(1) = 24 'hours
    minY(1) = 0
    maxY(1) = 180 'degrees

    XmajTick(1) = 4
    XminTick(1) = 2
    YmajTick(1) = 30
    YminTick(1) = 10

    'T ambient
    DiagramTitle(2) = "Ambient temperature"
    minX(2) = 0
    maxX(2) = 24 'hours
    minY(2) = -10
    maxY(2) = 60 'degrees

    XmajTick(2) = 4
    XminTick(2) = 2
    YmajTick(2) = 10
    YminTick(2) = 5

    'T surface
    DiagramTitle(3) = "Surface temperature"
    minX(3) = 0
    maxX(3) = 24 'hours
    minY(3) = -10
    maxY(3) = 60 'degrees

    XmajTick(3) = 4
    XminTick(3) = 2
    YmajTick(3) = 10
    YminTick(3) = 5

    'variation of temp on the external surface
    DiagramTitle(4) = "External temperature variation"
    minX(4) = 0
    maxX(4) = 180 'degrees
```

```
minY(4) = 0
maxY(4) = 100 'percentage

XmajTick(4) = 30
XminTick(4) = 10
YmajTick(4) = 10
YminTick(4) = 5

'variation of temp on the internal surface
DiagramTitle(5) = "Internal temperature variation"
minX(5) = 0
maxX(5) = 180 'degrees
minY(5) = 0
maxY(5) = 100 'percentage

XmajTick(5) = 30
XminTick(5) = 10
YmajTick(5) = 10
YminTick(5) = 5

'normalised creep
DiagramTitle(6) = "Normalised creep (t*)"
minX(6) = 0
maxX(6) = 100 'years
minY(6) = 0
maxY(6) = 0.0000040001 'that should be enough

XmajTick(6) = 10
XminTick(6) = 5
YmajTick(6) = 0.000001
YminTick(6) = 0.0000005

'normalised temperature
DiagramTitle(7) = "Normalised temperature (Phi)"
minX(7) = 0
maxX(7) = 80 'degrees
minY(7) = 0
maxY(7) = 100 'degrees

XmajTick(7) = 10
XminTick(7) = 5
YmajTick(7) = 10
YminTick(7) = 5

crMonthName(1) = "January"
```

```
    crMonthName(2) = "February"
    crMonthName(3) = "March"
    crMonthName(4) = "April"
    crMonthName(5) = "May"
    crMonthName(6) = "June"
    crMonthName(7) = "July"
    crMonthName(8) = "August"
    crMonthName(9) = "September"
    crMonthName(10) = "October"
    crMonthName(11) = "November"
    crMonthName(12) = "December"

    crDaysOfMonth(1) = 31
    crDaysOfMonth(2) = 28
    crDaysOfMonth(3) = 31
    crDaysOfMonth(4) = 30
    crDaysOfMonth(5) = 31
    crDaysOfMonth(6) = 30
    crDaysOfMonth(7) = 31
    crDaysOfMonth(8) = 31
    crDaysOfMonth(9) = 30
    crDaysOfMonth(10) = 31
    crDaysOfMonth(11) = 30
    crDaysOfMonth(12) = 31

End Sub

Public Function SaveDiagram(File As String, _
ByRef Diagram As DiagramInfo) As DiagramResults
On Error GoTo Er

    Dim myChan As Long
    Dim iA As Long

    SaveDiagram = OK

    myChan = FreeFile

    Open File For Output As myChan
    For iA = 0 To Diagram.Entries - 1
        Print #myChan, myStr(Diagram.Points(iA).X) & vbTab &
myStr(Diagram.Points(iA).Y)
    Next iA
    Close myChan
    Exit Function
```

```
Er:
SaveDiagram = ERROR_CRITICAL
Close myChan
End Function

Public Function LoadDiagram(File As String, _
ByRef Diagram As DiagramInfo) As DiagramResults
On Error GoTo Er

    Dim myChan As Long
    Dim sLine As String
    Dim splitLine As Variant

    Diagram.Entries = 0

    If Len(Dir(File)) = 0 Then
        LoadDiagram = ERROR_FILENOTFOUND
        Exit Function
    End If

    LoadDiagram = OK

    myChan = FreeFile

    Open File For Input As myChan

    Do While Not EOF(myChan)
        Line Input #myChan, sLine
        splitLine = Split(sLine, vbTab)

        If UBound(splitLine) <> 1 Then
            'dont include it in the fixed list
            LoadDiagram = OK_ERRORFIXED
        Else
            splitLine(0) = Replace(splitLine(0), ",", ".")
            splitLine(1) = Replace(splitLine(1), ",", ".")

            Diagram.Entries = Diagram.Entries + 1
            ReDim Preserve Diagram.Points(0 To Diagram.Entries - 1)
            Diagram.Points(Diagram.Entries - 1).X = Val(splitLine(0))
            Diagram.Points(Diagram.Entries - 1).Y = Val(splitLine(1))
        End If
    Loop
```

```
    Close myChan
    Exit Function

Er:
LoadDiagram = ERROR_CRITICAL
Close myChan
End Function

Public Function FixDiagram(ByRef Diagram As DiagramInfo, _
minX As Double, maxX As Double, _
minY As Double, maxY As Double) As DiagramResults

On Error GoTo Er

    Dim tmP As PointInfo
    Dim iA As Long
    Dim iB As Long

    If Diagram.Entries = 0 Then
        tmP.X = minX: tmP.Y = minY
        AddDiagramEntry Diagram, tmP
        tmP.X = maxX
        AddDiagramEntry Diagram, tmP
        FixDiagram = OK_ERRORFIXED
        Exit Function
    End If


    FixDiagram = OK

Restart:

    For iA = 0 To Diagram.Entries - 1
        If Diagram.Points(iA).X < minX Then
            DeleteDiagramEntry Diagram, iA
            FixDiagram = OK_ERRORFIXED
            GoTo Restart
        End If

        If Diagram.Points(iA).X > maxX Then
            DeleteDiagramEntry Diagram, iA
            FixDiagram = OK_ERRORFIXED
            GoTo Restart
        End If
```

```
        If Diagram.Points(iA).Y < minY Then
            DeleteDiagramEntry Diagram, iA
            FixDiagram = OK_ERRORFIXED
            GoTo Restart
        End If

        If Diagram.Points(iA).Y > maxY Then
            DeleteDiagramEntry Diagram, iA
            FixDiagram = OK_ERRORFIXED
            GoTo Restart
        End If
    Next iA

    For iA = 0 To Diagram.Entries - 2
        If Diagram.Points(iA).X > Diagram.Points(iA + 1).X Then
            FixDiagram = OK_ERRORFIXED
            Exit For
        End If
    Next iA

    For iA = 0 To Diagram.Entries - 1
        For iB = iA + 1 To Diagram.Entries - 1
            If Diagram.Points(iB).X < Diagram.Points(iA).X Then
                tmP = Diagram.Points(iB)
                Diagram.Points(iB) = Diagram.Points(iA)
                Diagram.Points(iA) = tmP
            End If
        Next iB
    Next iA

StartAgain:
    For iA = 0 To Diagram.Entries - 2
        If Diagram.Points(iA).X = Diagram.Points(iA + 1).X Then
            DeleteDiagramEntry Diagram, iA
            FixDiagram = OK_ERRORFIXED
            GoTo StartAgain
        End If
    Next iA

    If Diagram.Points(0).X <> minX Then
        tmP.X = minX: tmP.Y = minY
        AddDiagramEntry Diagram, tmP
    End If

    If Diagram.Points(Diagram.Entries - 1).X <> maxX Then
```

```
        tmP.X = maxX: tmP.Y = minY
        AddDiagramEntry Diagram, tmP
    End If

    Exit Function

Er:
FixDiagram = ERROR_CRITICAL
End Function

Public Sub AddDiagramEntry(ByRef Diagram As DiagramInfo, Point As PointInfo)
On Error Resume Next

    'we assume that the points are in an ascending order
    'no check whether the point is valid!
Dim iA As Long
Dim Up As Long
Dim Dn As Long
Dim Cur As Long

    If Diagram.Entries = 0 Then
            ReDim Preserve Diagram.Points(0)
            Diagram.Points(0) = Point
            Diagram.Entries = 1
            Exit Sub
    End If

Dn = 0: Up = Diagram.Entries - 1

    If Diagram.Points(Dn).X = Point.X Then
        Diagram.Points(Dn).Y = Point.Y
        Exit Sub
    ElseIf Diagram.Points(Up).X = Point.X Then
        Diagram.Points(Up).Y = Point.Y
        Exit Sub
    ElseIf Diagram.Points(Dn).X > Point.X Then
            ReDim Preserve Diagram.Points(0 To Diagram.Entries)
            For iA = Diagram.Entries To 1 Step -1
                Diagram.Points(iA) = Diagram.Points(iA - 1)
            Next iA
            Diagram.Points(0) = Point
            Diagram.Entries = Diagram.Entries + 1
            Exit Sub
    ElseIf Diagram.Points(Up).X < Point.X Then
            ReDim Preserve Diagram.Points(0 To Diagram.Entries)
```

```
            Diagram.Points(Diagram.Entries) = Point
            Diagram.Entries = Diagram.Entries + 1
            Exit Sub
    End If

Do
    Cur = Fix((Dn + Up) / 2)

    If Diagram.Points(Cur).X = Point.X Then
        'the point already exists, change the value and exit
        Diagram.Points(Cur).Y = Point.Y
        Exit Sub
    ElseIf Diagram.Points(Cur).X > Point.X Then
        'the point is between dn and cur
        Up = Cur
    Else
        'the point is between cur and up
        Dn = Cur
    End If

Loop While Up > Dn + 1

    ReDim Preserve Diagram.Points(0 To Diagram.Entries)

    For iA = Diagram.Entries To Up + 1 Step -1
        Diagram.Points(iA) = Diagram.Points(iA - 1)
    Next iA
        Diagram.Points(Up) = Point
        Diagram.Entries = Diagram.Entries + 1

End Sub

Public Sub DeleteDiagramEntry(ByRef Diagram As DiagramInfo, Entry As Long)
On Error Resume Next
    Dim iA As Long

    For iA = Entry To Diagram.Entries - 2
        Diagram.Points(iA) = Diagram.Points(iA + 1)
    Next iA

    Diagram.Entries = Diagram.Entries - 1
    ReDim Preserve Diagram.Points(0 To Diagram.Entries - 1) As PointInfo

End Sub
```

```
Public Function Interpolate(ByRef Diagram As DiagramInfo, Point As
PointInfo) As DiagramResults
On Error Resume Next

    'we assume that the points are in an ascending order
    'no check whether the point is valid!

Dim iA As Long
Dim Up As Long
Dim Dn As Long
Dim Cur As Long

Interpolate = ERROR_CRITICAL

    If Diagram.Entries = 0 Then Exit Function

    If Diagram.Points(0).X > Point.X Or Diagram.Points(Diagram.Entries -
1).X < Point.X Then Exit Function

    Dn = 0: Up = Diagram.Entries - 1

Do
    Cur = Fix((Dn + Up) / 2)

    If Diagram.Points(Dn).X = Point.X Then
        Interpolate = OK
        Point.Y = Diagram.Points(Dn).Y
        Exit Function
    ElseIf Diagram.Points(Up).X = Point.X Then
        Interpolate = OK
        Point.Y = Diagram.Points(Up).Y
        Exit Function
    ElseIf Diagram.Points(Cur).X = Point.X Then
        Interpolate = OK
        Point.Y = Diagram.Points(Cur).Y
        Exit Function
    ElseIf Diagram.Points(Cur).X > Point.X Then
        Up = Cur 'the point is between dn and cur
    Else
        Dn = Cur 'the point is between cur and up
    End If

Loop While Up > Dn + 1

Point.Y = Diagram.Points(Dn).Y + _
```

```
(Diagram.Points(Up).Y - Diagram.Points(Dn).Y) * (Point.X -
Diagram.Points(Dn).X) _
/ (Diagram.Points(Up).X - Diagram.Points(Dn).X)

    Interpolate = OK
    Exit Function
Er:
End Function
```

modCalculations.bas:

```
Option Explicit

Public NumOfRings As Long
Public NumOfSectors As Long
Public NumOfSlices As Long

Public gdColHeight As Double
Public gdIRadius As Double
Public gdERadius As Double

Public gdUnitLoad As Double
Public gdEModulus As Double
Public gdCompForce As Double

Public gdRefTemp As Double
Public gdAThermal As Double

Public gdTime As Double
Public gdTimeInterval As Double
Public glDaysElapsed As Long

Public BAngle() As Double
Public BArea() As Double
Public BPosition() As PointInfo
Public BTemperature() As Double

Public gdSunPosition As Double
Public gdAmbientTemperature As Double
Public gdSurfaceTemperature As Double

Public BAngleToSun() As Double
Public BExtTemperature() As Double
Public BIntTemperature() As Double
```

```
Public EFreeThermal() As Double
Public DeltaE() As Double
Public Ezero() As Double
Public Kx() As Double
Public Ky() As Double
Public Fz() As Double
Public gdConcArea As Double
Public gdConcI As Double 'the exact value

Public SlopeX() As Double 'around the X axis, leads to y-deflection
Public SlopeY() As Double 'around the Y axis, leads to x-deflection

Public Type DeflectioN_Info
    DayCalculated As String
    DefX() As Double
    DefY() As Double
    DefZ() As Double
    TipX As Double
    TipY As Double
    TipZ As Double
End Type

Public DeflectionS() As DeflectioN_Info
Public DeflectionSCounter As Long

Public DefX() As Double
Public DefY() As Double
Public DefZ() As Double
Public TipX As Double
Public TipY As Double
Public TipZ As Double

Public gdConcIxx As Double 'the values calculated from the blocks
Public gdConcIyy As Double

Public MsGCounter As Long
Public MsG() As String
Public Const Pi As Double = 3.14159265358979
Public Const HoursPerYear As Double = 8766

Public Sub CalcInitialiseDeflections()
On Error Resume Next
    DeflectionSCounter = 0
    ReDim DeflectionS(0) As DeflectioN_Info
```

```
End Sub

Public Sub CalcStoreDeflections(ByVal DayCalculated As String)
'This sub stores the deflections
On Error Resume Next

    DeflectionSCounter = DeflectionSCounter + 1

    ReDim Preserve DeflectionS(0 To DeflectionSCounter) As DeflectioN_Info

    DeflectionS(DeflectionSCounter).DayCalculated = DayCalculated
    DeflectionS(DeflectionSCounter).DefX = DefX
    DeflectionS(DeflectionSCounter).DefY = DefY
    DeflectionS(DeflectionSCounter).DefZ = DefZ
    DeflectionS(DeflectionSCounter).TipX = TipX
    DeflectionS(DeflectionSCounter).TipY = TipY
    DeflectionS(DeflectionSCounter).TipZ = TipZ

End Sub

Public Sub CalcDeflections()
'This sub calculates the slopes and deflections in the X,Y,Z direction along
the height
'and stores them in the public variables DefX(), DefY(), DefZ() which are
previously erased.
'For convenience, the variables TipX, TipY, TipZ are also calculated here.
'Data needed: (for each slice of the pylon)
    'Ezero()
    'Kx()
    'Ky()
'Data Calculated:
    'DefX(), TipX
    'DefY(), TipY
    'DefZ(), TipZ
    'SlopeX(), SlopeY()

On Error Resume Next
    Dim iC As Long
    Dim dH As Double

    dH = gdColHeight / NumOfSlices

ReDim DefZ(0 To NumOfSlices)
    For iC = 1 To NumOfSlices
        DefZ(iC) = DefZ(iC - 1) - Ezero(iC) * dH 'because z axis extends
downwards
```

```
    Next iC
TipZ = DefZ(NumOfSlices)

ReDim SlopeX(0 To NumOfSlices) As Double 'around the X axis, causes Y
deflection
    For iC = 1 To NumOfSlices
'        SlopeX(iC) = SlopeX(iC - 1) + 0.5 * (Kx(iC - 1) + Kx(iC)) * dh
        'if we use this, the results are identical to these of VW:
        SlopeX(iC) = SlopeX(iC - 1) + Kx(iC) * dH
    Next iC

ReDim DefY(0 To NumOfSlices)
    For iC = 1 To NumOfSlices
        'trapezoidal rule
        DefY(iC) = DefY(iC - 1) - 0.5 * (SlopeX(iC - 1) + SlopeX(iC)) * dH
    Next iC
TipY = DefY(NumOfSlices)

ReDim SlopeY(0 To NumOfSlices) As Double 'around the Y axis, causes X
deflection
    For iC = 1 To NumOfSlices
'        SlopeY(iC) = SlopeY(iC - 1) + 0.5 * (Ky(iC - 1) + Ky(iC)) * dh
        SlopeY(iC) = SlopeY(iC - 1) + Ky(iC) * dH
    Next iC

ReDim DefX(0 To NumOfSlices)
    For iC = 1 To NumOfSlices
        DefX(iC) = DefX(iC - 1) - 0.5 * (SlopeY(iC - 1) + SlopeY(iC)) * dH
    Next iC
TipX = DefX(NumOfSlices)

End Sub

Public Sub CalcVerifyEquations()
'This sub verifies the validity of the calculations of Ezero, Kx, Ky
'Each time it is called, the Msg() is reinitialised
'Msg() is filled with strings whenever the force or moment equilibrium
'accuracy is less than 1N or 1Nm respectively
'That should be enough!

On Error Resume Next
    Dim iA As Long
    Dim iB As Long
    Dim iC As Long
    Dim dTmP As Double
```

```
    Dim tmP As Double
    Const AkriB As Double = 0.000001

    MsGCounter = 0

'check Eo
For iC = 1 To NumOfSlices
    dTmP = 0
        For iA = 1 To NumOfRings
            For iB = 1 To NumOfSectors
                dTmP = dTmP + ((EFreeThermal(iA, iB) - DeltaE(iA, iB, iC)) -
Ezero(iC) - Ky(iC) * BPosition(iA, iB).X - Kx(iC) * BPosition(iA, iB).Y) *
BArea(iA)
            Next iB
        Next iA
    dTmP = dTmP * gdEModulus * 1000
    If Abs(dTmP - Fz(iC)) > AkriB Then
        MsGCounter = MsGCounter + 1
        ReDim Preserve MsG(1 To MsGCounter) As String
        MsG(MsGCounter) = "Caution! FE, Slice :" & Str$(iC) & " , LHS =" &
myStr(Fz(iC)) & " , RHS =" & myStr(dTmP)
    End If
Next iC

'check moment equilibrium around x axis
For iC = 1 To NumOfSlices
    dTmP = 0
        For iA = 1 To NumOfRings
            For iB = 1 To NumOfSectors
                dTmP = dTmP + ((EFreeThermal(iA, iB) - DeltaE(iA, iB, iC)) -
Ezero(iC) - Ky(iC) * BPosition(iA, iB).X - Kx(iC) * BPosition(iA, iB).Y) *
BArea(iA) * BPosition(iA, iB).Y
            Next iB
        Next iA
    dTmP = dTmP * gdEModulus * 1000
    If Abs(dTmP) > AkriB Then
        MsGCounter = MsGCounter + 1
        ReDim Preserve MsG(1 To MsGCounter) As String
        MsG(MsGCounter) = "Caution! X-ME, Slice :" & Str$(iC) & " , LHS =0 ,
RHS =" & myStr(dTmP)
    End If
Next iC

'check moment equilibrium around y axis
For iC = 1 To NumOfSlices
    dTmP = 0
        For iA = 1 To NumOfRings
```

```
            For iB = 1 To NumOfSectors
                dTmP = dTmP + ((EFreeThermal(iA, iB) - DeltaE(iA, iB, iC)) -
Ezero(iC) - Ky(iC) * BPosition(iA, iB).X - Kx(iC) * BPosition(iA, iB).Y) *
BArea(iA) * BPosition(iA, iB).X
            Next iB
        Next iA
    dTmP = dTmP * gdEModulus * 1000
    If Abs(dTmP) > AkriB Then
        MsGCounter = MsGCounter + 1
        ReDim Preserve MsG(1 To MsGCounter) As String
        MsG(MsGCounter) = "Caution! Y-ME, Slice :" & Str$(iC) & " , LHS =0 ,
RHS =" & myStr(dTmP)
    End If
Next iC

End Sub

Public Sub CalcVerifyDeflections()
'This sub verifies the validity of the deflection calculations
'against the Virtual Work method

On Error Resume Next
    Dim iA As Long
    Dim iB As Long
    Dim iC As Long
    Dim dH As Double
    Dim tmP As Double

Debug.Print

dH = gdColHeight / NumOfSlices

tmP = 0
For iC = 1 To NumOfSlices
    tmP = tmP - Kx(iC) * dH * (NumOfSlices - iC + 0.5) * dH
Next iC

Debug.Print "Virtual Work Y=" & Str$(tmP) & " , TipY=" & Str$(TipY) & " ,
Error=" & myStr(FNRoundNumber(100 * (tmP - TipY) / TipY, 5)) & "%"

tmP = 0
For iC = 1 To NumOfSlices
    tmP = tmP - Ky(iC) * dH * (NumOfSlices - iC + 0.5) * dH
Next iC

Debug.Print "Virtual Work X=" & Str$(tmP) & " , TipX=" & Str$(TipX) & " ,
Error=" & myStr(FNRoundNumber(100 * (tmP - TipX) / TipX, 5)) & "%"
```

```
End Sub


Public Sub CalcCurvatures()
'This sub calculates the thermoelastic/creep curvatures based on the
temperatures of each block
'Data needed:
    'EFreeThermal()
    'DeltaE() -> set to zero for a simple thermoelastic solution
    'BArea()
    'BPosition()
    'Ixx, Iyy
'Data Calculated:
    'Ezero()
    'Kx()
    'Ky()
On Error Resume Next
    Dim iA As Long
    Dim iB As Long
    Dim iC As Long

    Dim dTmP As Double
    Dim TmP1 As Double
    Dim TmP2 As Double

    'now calculate Eo
    ReDim Ezero(1 To NumOfSlices) As Double


For iC = 1 To NumOfSlices
    'dtmp = SS(Ef dA)
    dTmP = 0
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            dTmP = dTmP + (EFreeThermal(iA, iB) - DeltaE(iA, iB, iC)) *
BArea(iA)
        Next iB
    Next iA


    Ezero(iC) = (dTmP - Fz(iC) / (1000 * gdEModulus)) / gdConcArea
Next iC


    'now calculate Kx,Ky


    ReDim Kx(0 To NumOfSlices) As Double 'reserve 0 for the numerical
integration
    ReDim Ky(0 To NumOfSlices) As Double
```

```
For iC = 1 To NumOfSlices

    'tmp1=SS(y*ef*dA)
    TmP1 = 0
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            TmP1 = TmP1 + BPosition(iA, iB).Y * (EFreeThermal(iA, iB) -
DeltaE(iA, iB, iC)) * BArea(iA)
        Next iB
    Next iA

    'tmp2=SS(x*ef*dA)
    TmP2 = 0
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            TmP2 = TmP2 + BPosition(iA, iB).X * (EFreeThermal(iA, iB) -
DeltaE(iA, iB, iC)) * BArea(iA)
        Next iB
    Next iA

    Kx(iC) = TmP1 / gdConcIxx
    Ky(iC) = TmP2 / gdConcIyy
Next iC

End Sub

Public Sub CalcCreepCurvatures()
'This sub calculates the creep curvatures based on the DeltaE() alone
'The slices don't satisfy equilibrium! we are interested only in creep
strains
'Data needed:
    'DeltaE()
    'BArea()
    'BPosition()
    'Ixx, Iyy
'Data Calculated:
    'Ezero()
    'Kx()
    'Ky()
On Error Resume Next
    Dim iA As Long
    Dim iB As Long
    Dim iC As Long

    Dim dTmP As Double
```

```
    Dim TmP1 As Double
    Dim TmP2 As Double

    'now calculate Eo
    ReDim Ezero(1 To NumOfSlices) As Double

For iC = 1 To NumOfSlices
    'dtmp = SS(Ef dA)
    dTmP = 0
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            dTmP = dTmP - DeltaE(iA, iB, iC) * BArea(iA)
        Next iB
    Next iA

    Ezero(iC) = dTmP / gdConcArea
Next iC

    'now calculate Kx,Ky

    ReDim Kx(0 To NumOfSlices) As Double 'reserve 0 for the numerical
integration
    ReDim Ky(0 To NumOfSlices) As Double

For iC = 1 To NumOfSlices

    'tmp1=SS(y*ef*dA)
    TmP1 = 0
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            TmP1 = TmP1 + BPosition(iA, iB).Y * (-DeltaE(iA, iB, iC)) *
BArea(iA)
        Next iB
    Next iA

    'tmp2=SS(x*ef*dA)
    TmP2 = 0
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            TmP2 = TmP2 + BPosition(iA, iB).X * (-DeltaE(iA, iB, iC)) *
BArea(iA)
        Next iB
    Next iA

    Kx(iC) = TmP1 / gdConcIxx
    Ky(iC) = TmP2 / gdConcIyy
```

```
Next iC


End Sub
Public Sub CalcCreepStrains()
'This sub calculates the creep strains based on the temperatures of each
block
'Data needed:
    'EFreeThermal()
    'Ezero(), Kx(), Ky()
    'BArea()
    'BPosition()
'Data Updated:
    'DeltaE()
On Error Resume Next
    Dim iA As Long
    Dim iB As Long
    Dim iC As Long


    Dim mPtSt As PointInfo
    Dim mPtEn As PointInfo
    Dim ReS As DiagramResults
    Dim DtStar As Double
    Dim dTmP As Double


    Dim Phi() As Double
    ReDim Phi(1 To NumOfRings, 1 To NumOfSectors) As Double


    'calculate dtstar (common for all blocks)
    mPtSt.X = (24 * (glDaysElapsed - 1) + gdTime - gdTimeInterval / 2) /
HoursPerYear
        ReS = Interpolate(Diagram(6), mPtSt)
    mPtEn.X = (24 * (glDaysElapsed - 1) + gdTime + gdTimeInterval / 2) /
HoursPerYear
        ReS = Interpolate(Diagram(6), mPtEn)
    DtStar = mPtEn.Y - mPtSt.Y


    'Debug.Print "Years :" & Str$(mPtEn.X)


    'now calculate the phi(T) for each block (common for all slices)
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            mPtSt.X = BTemperature(iA, iB)
            ReS = Interpolate(Diagram(7), mPtSt)
            Phi(iA, iB) = mPtSt.Y
        Next iB
    Next iA
```

```
    'finally calculate the creep strains and add them to DeltaE()
For iC = 1 To NumOfSlices
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            dTmP = ((EFreeThermal(iA, iB) - DeltaE(iA, iB, iC)) - _
                Ezero(iC) - Ky(iC) * BPosition(iA, iB).X - _
                    Kx(iC) * BPosition(iA, iB).Y) * _
                    (1000 * gdEModulus) * Phi(iA, iB) * DtStar

            DeltaE(iA, iB, iC) = DeltaE(iA, iB, iC) + dTmP
        Next iB
    Next iA
Next iC

End Sub

Public Sub CalcFreeStrains()
'This sub must be executed after the calculation of the temperatures for
each block.
'This sub must be executed for both the thermoelastic and creep solutions.
'Data needed:
    'BTemperature()
'Data Calculated:
    'EFreeThermal()

On Error Resume Next
    Dim iA As Long
    Dim iB As Long

    ReDim EFreeThermal(1 To NumOfRings, 1 To NumOfSectors) As Double

    'calculate the free thermal strains
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            EFreeThermal(iA, iB) = gdAThermal * (BTemperature(iA, iB) -
gdRefTemp)
        Next iB
    Next iA

End Sub

Public Sub CalcInitialiseDeltaE()
    'erase the creep strains from previous solutions
    ReDim DeltaE(1 To NumOfRings, 1 To NumOfSectors, 1 To NumOfSlices) As
Double
```

```
End Sub

Public Sub CalcInitialiseEFreeThermal()
    'erase the free thermal strains from previous solutions
    ReDim EFreeThermal(1 To NumOfRings, 1 To NumOfSectors) As Double
End Sub

Public Sub CalcInitialise()
'This sub must be executed after the calculation of the temperatures for
each block.
'This sub must be executed for both the thermoelastic and creep solutions.
'It also initialises (sets to zero) the strains due to creep
'(DeltaE()) from previous calculations
'Data needed:
    'BPosition()
'Data Calculated:
    'gdConcIxx, gdConcIyy (needed for calculating Ezero, Kx, Ky
    'Fz() -> compessive force for each slice

On Error Resume Next
    Dim iA As Long
    Dim iB As Long
    Dim iC As Long

    Dim dTmP As Double

    'now calculate the compression force for each block
    ReDim Fz(1 To NumOfSlices) As Double

    dTmP = gdConcArea * (gdColHeight / NumOfSlices) * gdUnitLoad / 1000
'force in MN

    For iC = 1 To NumOfSlices
        Fz(iC) = gdCompForce + (NumOfSlices - iC) * dTmP
    Next iC

    'Second moment of area, calculated from the blocks (not the exact
values!)
    gdConcIxx = 0
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            gdConcIxx = gdConcIxx + BPosition(iA, iB).Y ^ 2 * BArea(iA)
        Next iB
    Next iA

    gdConcIyy = 0
```

```
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            gdConcIyy = gdConcIyy + BPosition(iA, iB).X ^ 2 * BArea(iA)
        Next iB
    Next iA

End Sub

Public Sub CalcTemperatures()
'This sub calculates the thermoelastic solution based on the temperatures of
each block
'Data needed:
    'gdTime
    'BAngle()
    'BArea()
    'BPosition()
'Data Calculated:
    'gdSunPosition
    'gdAmbientTemperature
    'gdSurfaceTemperature
    'BAngleToSun()
    'BIntTemperature()
    'BExtTemperature()
    'BTemperature()

On Error Resume Next
Dim tmP As PointInfo
Dim ReS As DiagramResults
Dim iA As Long
Dim iB As Long
Dim dTmP As Double
Dim dRad As Double
Dim dIntT As Double
Dim dExtT As Double

tmP.X = gdTime

    ReS = Interpolate(Diagram(1), tmP)
    If ReS <> OK Then
        'debug.Print "CalcTemperatures: Error in interpolation 1"
    End If
    gdSunPosition = tmP.Y

    ReS = Interpolate(Diagram(2), tmP)
    If ReS <> OK Then
```

```
        'debug.Print "CalcTemperatures: Error in interpolation 2"
    End If
    gdAmbientTemperature = tmP.Y

    ReS = Interpolate(Diagram(3), tmP)
    If ReS <> OK Then
        'debug.Print "CalcTemperatures: Error in interpolation 3"
    End If
    gdSurfaceTemperature = tmP.Y

    ReDim BAngleToSun(1 To NumOfSectors) As Double

    For iA = 1 To NumOfSectors
        BAngleToSun(iA) = BAngle(iA) - FNDegreesToRad(gdSunPosition)
        If BAngleToSun(iA) > Pi Then BAngleToSun(iA) = BAngleToSun(iA) - 2 *
Pi
    Next iA

    ReDim BExtTemperature(1 To NumOfSectors) As Double

    For iA = 1 To NumOfSectors
        tmP.X = FNRadToDegrees(Abs(BAngleToSun(iA)))
        ReS = Interpolate(Diagram(4), tmP)
        If ReS <> OK Then
            'debug.Print "CalcTemperatures: Error in interpolation 4"
        End If
        BExtTemperature(iA) = tmP.Y
    Next iA

    ReDim BIntTemperature(1 To NumOfSectors) As Double

    For iA = 1 To NumOfSectors
        tmP.X = FNRadToDegrees(Abs(BAngleToSun(iA)))
        ReS = Interpolate(Diagram(5), tmP)
        If ReS <> OK Then
            'debug.Print "CalcTemperatures: Error in interpolation 5"
        End If
        BIntTemperature(iA) = tmP.Y
    Next iA

    ReDim BTemperature(1 To NumOfRings, 1 To NumOfSectors) As Double

    dTmP = (gdERadius - gdIRadius) / NumOfRings

    For iB = 1 To NumOfSectors
```

```
        dIntT = gdAmbientTemperature + 0.01 * BIntTemperature(iB) *
(gdSurfaceTemperature - gdAmbientTemperature)
        dExtT = gdAmbientTemperature + 0.01 * BExtTemperature(iB) *
(gdSurfaceTemperature - gdAmbientTemperature)
            For iA = 1 To NumOfRings
                dRad = (iA - 0.5) * dTmP
                BTemperature(iA, iB) = dIntT + (dExtT - dIntT) * (dRad /
(gdERadius - gdIRadius))
            Next iA
    Next iB

End Sub

Public Sub CalcGeometry()
'This sub must be executed prior to any calculations.
'Data needed:
    '- (just the global variables, dimensions of the pylon etc, which are
assumed to be ready for use
'Data Calculated:
    'BAngle()
    'BArea()
    'BPosition()
    'gdConcArea, gdConcI (the exact values!)
On Error Resume Next
Dim dTmP As Double
Dim diRad As Double
Dim deRad As Double

Dim iA As Long
Dim iB As Long

    'angle corresponding to each sector
    ReDim BAngle(1 To NumOfSectors) As Double
    dTmP = 2 * Pi / NumOfSectors

    For iA = 1 To NumOfSectors
        BAngle(iA) = dTmP * (iA - 0.5)
    Next iA

    'area of each block
    ReDim BArea(1 To NumOfRings) As Double
    dTmP = (gdERadius - gdIRadius) / NumOfRings

    For iA = 1 To NumOfRings
        diRad = gdIRadius + (iA - 1) * dTmP
        deRad = diRad + dTmP
```

```
        BArea(iA) = Pi * (deRad ^ 2 - diRad ^ 2) / (NumOfSectors)
    Next iA

    'position of each block
    ReDim BPosition(1 To NumOfRings, 1 To NumOfSectors) As PointInfo
    dTmP = (gdERadius - gdIRadius) / NumOfRings

    For iB = 1 To NumOfSectors
        For iA = 1 To NumOfRings
            diRad = gdIRadius + (iA - 0.5) * dTmP 'now this is the block's
radius
            BPosition(iA, iB).X = diRad * Cos(BAngle(iB))
            BPosition(iA, iB).Y = diRad * Sin(BAngle(iB))
        Next iA
    Next iB

    'concrete area
    gdConcArea = Pi * (gdERadius ^ 2 - gdIRadius ^ 2)

    'second moment of area
    gdConcI = Pi * (gdERadius ^ 4 - gdIRadius ^ 4) / 4



End Sub

Public Function FNRadToDegrees(ByVal Rad As Double) As Double
    FNRadToDegrees = Rad * 180 / Pi
End Function

Public Function FNDegreesToRad(ByVal Degrees As Double) As Double
    FNDegreesToRad = Degrees * Pi / 180
End Function

Public Function FNRoundNumber(ByVal Number As Variant, Optional ByVal DecS
As Long = 0) As Double
    Dim tmP As Double, DecShift As Long, ProS As Long
    ProS = Sgn(CDbl(Number)) 'sign
    tmP = ProS * CDbl(Number)
    DecShift = 10 ^ DecS
    FNRoundNumber = ProS * (Fix((tmP + 0.5 / DecShift) * DecShift)) /
DecShift
End Function

Public Function FNDaysOfMonth(ByVal Month As Long, ByVal Year As Long) As
Long
On Error Resume Next
```

```
    If FNIsLeapYear(Year) And Month = 2 Then
        FNDaysOfMonth = 29
    Else
        FNDaysOfMonth = crDaysOfMonth(Month)
    End If
End Function


Public Function FNDate(ByVal Day As Long, ByVal Month As Long, ByVal Year As
Long) As String
On Error Resume Next
    FNDate = Trim$(Str$(Day)) & "/" & Trim$(Str$(Month)) & "/" &
Trim$(Str$(Year))
End Function


Public Function FNIsLeapYear(Year As Long) As Boolean
    If (Year Mod 4 = 0) And ((Year Mod 100 <> 0) Or (Year Mod 400 = 0)) Then
        FNIsLeapYear = True
    Else
        FNIsLeapYear = False
    End If
End Function
```

frmThermoElastic.frm/cmdCalculate_click()

```
Private Sub cmdCalculate_Click()
On Error Resume Next

Dim iA As Long
Dim iB As Long
Dim iC As Long


    If Val(txtHour) < 0 Or Val(txtHour) > 24 Then
        MsgBox "The hour you selected is invalid.", vbOKOnly +
vbExclamation, "Error"
        Exit Sub
    End If


    For iA = 1 To 5
        If cboDiagram(iA).ListIndex < 0 Then
            MsgBox "You haven't selected the " & DiagramTitle(iA) & "
diagram.", vbOKOnly + vbExclamation, "Error"
            Exit Sub
        End If
    Next iA


    NumOfRings = Val(frmMain.txtI)
```

```
    NumOfSectors = Val(frmMain.txtJ)
    NumOfSlices = Val(frmMain.txtK)

    gdColHeight = Val(frmMain.txtHeight)
    gdIRadius = Val(frmMain.txtIRadius)
    gdERadius = Val(frmMain.txtERadius)

    gdCompForce = Val(frmMain.txtF)
    gdEModulus = Val(frmMain.txtE)
    gdUnitLoad = Val(frmMain.txtUnitLoad)

    gdRefTemp = Val(frmMain.txtRefTemp)
    gdAThermal = Val(frmMain.txtAThermal)

    gdTime = Val(txtHour)



        lstResults.Clear
        lstResults.AddItem ">> Main Data..."
        lstResults.AddItem "Time :" & myStr(gdTime) & " hours."
        lstResults.AddItem "Column Height :" & myStr(gdColHeight) & " m."
        lstResults.AddItem "Internal Radius :" & myStr(gdIRadius) & " m."
        lstResults.AddItem "External Radius :" & myStr(gdERadius) & " m."

        lstResults.AddItem "Compression Force :" & myStr(gdCompForce) & "
MN."
        lstResults.AddItem "Elastic Modulus :" & myStr(gdEModulus) & " GPa."
        lstResults.AddItem "Unit Load :" & myStr(gdUnitLoad) & " KN/m³."
        lstResults.AddItem "Reference Temperature :" & myStr(gdRefTemp) & "
oC."
        lstResults.AddItem "Coefficient of thermal expansion :" &
myStr(gdAThermal) & " 1/oC."

        lstResults.AddItem "Number Of Rings :" & myStr(NumOfRings)
        lstResults.AddItem "Number Of Sectors :" & myStr(NumOfSectors)
        lstResults.AddItem "Number Of Slices :" & myStr(NumOfSlices)

        lstResults.AddItem ">> Calculating Geometry..."
    CalcGeometry

    For iA = 1 To NumOfRings
        lstResults.AddItem "Ring" & Str$(iA) & " , Block Area :" &
myStr(BArea(iA)) & " m²."
    Next iA

    For iA = 1 To NumOfSectors
```

```
        lstResults.AddItem "Sector" & Str$(iA) & " , Angle :" &
myStr(FNRadToDegrees(BAngle(iA))) & " degrees."
    Next iA


    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            lstResults.AddItem "Block (" & Str$(iA) & "," & Str$(iB) & " ) ,
X :" & myStr(BPosition(iA, iB).X) & " m , Y :" & myStr(BPosition(iA, iB).Y)
& " m."
        Next iB
    Next iA


        lstResults.AddItem ">> Preliminary Calculations..."
    CalcInitialise
    lstResults.AddItem "Concrete Area :" & myStr(gdConcArea) & " m²."
    lstResults.AddItem "Second Moment of Area (exact value) :" &
myStr(gdConcI) & " m^4."
    lstResults.AddItem "Second Moment of Area (approx. value) :" &
myStr(gdConcIxx) & " m^4."


        lstResults.AddItem ">> Calculating Temperatures..."
    CalcTemperatures
    lstResults.AddItem "Sun Position :" & myStr(gdSunPosition) & " degrees."
    lstResults.AddItem "Ambient Temperature :" & myStr(gdAmbientTemperature)
& " degrees."
    lstResults.AddItem "Surface Temperature :" & myStr(gdSurfaceTemperature)
& " degrees."


    For iA = 1 To NumOfSectors
        lstResults.AddItem "Sector" & Str$(iA) & " , External Variation :" &
myStr(BExtTemperature(iA)) & " %."
    Next iA


    For iA = 1 To NumOfSectors
        lstResults.AddItem "Sector" & Str$(iA) & " , Internal Variation :" &
myStr(BIntTemperature(iA)) & " %."
    Next iA


    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            lstResults.AddItem "Block (" & Str$(iA) & "," & Str$(iB) & " ) ,
Temperature :" & myStr(BTemperature(iA, iB)) & " degrees."
        Next iB
    Next iA


        lstResults.AddItem ">> Calculating Free Thermal Strains..."
    CalcFreeStrains
```

```
    For iA = 1 To NumOfRings
        For iB = 1 To NumOfSectors
            lstResults.AddItem "Block (" & Str$(iA) & "," & Str$(iB) & " ) ,
Initial Free Strains :" & myStr(EFreeThermal(iA, iB))
        Next iB
    Next iA

    CalcInitialiseDeltaE

        lstResults.AddItem ">> Calculating Curvatures..."
    CalcCurvatures



    For iC = 1 To NumOfSlices
        lstResults.AddItem "Slice (" & Str$(iC) & " ) , Eo :" &
myStr(Ezero(iC))
    Next iC

    For iC = 1 To NumOfSlices
        lstResults.AddItem "Slice (" & Str$(iC) & " ) , Kx :" &
myStr(Kx(iC))
    Next iC

    For iC = 1 To NumOfSlices
        lstResults.AddItem "Slice (" & Str$(iC) & " ) , Ky :" &
myStr(Ky(iC))
    Next iC

        lstResults.AddItem ">> Verifying Equation Accuracy..."
    CalcVerifyEquations

    If MsGCounter > 0 Then
        lstResults.AddItem ">> Equation accuracy errors detected :"
        For iC = 1 To MsGCounter
            lstResults.AddItem MsG(iC)
        Next iC
    End If

        lstResults.AddItem ">> Calculating Deflections..."
    CalcDeflections

    lstResults.AddItem "X - Deflection :" & myStr(TipX) & " m."
    lstResults.AddItem "Y - Deflection :" & myStr(TipY) & " m."
    lstResults.AddItem "Z - Deflection :" & myStr(TipZ) & " m."

    'CalcVerifyDeflections 'enable this to compare the tip deflections to
those of Virtual Work
```

```
    cmdDrawSection.Enabled = True
    cmdSaveReport.Enabled = True

End Sub
```

frmCreep.frm/cmdCalculate_click()

```
Private Sub cmdCalculate_Click()
On Error Resume Next

    Dim iA As Long
    Dim iC As Long
    Dim dA As Double

    Dim StDay As Long
    Dim StMonth As Long
    Dim StYear As Long
    Dim EnDay As Long
    Dim EnMonth As Long
    Dim EnYear As Long
    Dim CurDay As Long
    Dim DaysOfCurMonth As Long
    Dim CurMonth As Long
    Dim CurYear As Long
    Dim NumOfStoredDef As Long
    Dim StoreDefEvery As Long
    Dim StoreDefAt As Double

    Dim STimer As Single
    Dim ETimer As Single
    Dim SDate As Date
    Dim EDate As Date
    Dim SecS As Long
    Dim strTime As String

    For iA = 1 To 5
        'this means that there's no diagram in the library
        'in every other case, there will always be a diagram associated to
each and every month
        If cboDiagram(iA).ListCount = 0 Then
            MsgBox "The " & DiagramTitle(iA) & " diagram is not valid.",
vbOKOnly + vbExclamation, "Error"
            Exit Sub
        End If
```

```
    Next iA

    If cboTimeInterval.ListIndex < 0 Then
        MsgBox "You haven't set the time interval.", vbOKOnly +
vbExclamation, "Error"
        Exit Sub
    End If

    If cboStDay.ListIndex < 0 Then
        MsgBox "You haven't set the starting day.", vbOKOnly +
vbExclamation, "Error"
        Exit Sub
    End If

    If cboStMonth.ListIndex < 0 Then
        MsgBox "You haven't set the starting month.", vbOKOnly +
vbExclamation, "Error"
        Exit Sub
    End If

    If cboStYear.ListIndex < 0 Then
        MsgBox "You haven't set the starting year.", vbOKOnly +
vbExclamation, "Error"
        Exit Sub
    End If

    If cboEnDay.ListIndex < 0 Then
        MsgBox "You haven't set the ending day.", vbOKOnly + vbExclamation,
"Error"
        Exit Sub
    End If

    If cboEnMonth.ListIndex < 0 Then
        MsgBox "You haven't set the ending month.", vbOKOnly +
vbExclamation, "Error"
        Exit Sub
    End If

    If cboEnYear.ListIndex < 0 Then
        MsgBox "You haven't set the ending year.", vbOKOnly + vbExclamation,
"Error"
        Exit Sub
    End If

    If cboEnYear.ListIndex < cboStYear.ListIndex Then
        MsgBox "The ending date cannot be earlier than the starting date.",
vbOKOnly + vbExclamation, "Error"
        Exit Sub
```

```
    ElseIf cboEnYear.ListIndex = cboStYear.ListIndex And _
        cboEnMonth.ListIndex < cboStMonth.ListIndex Then
        MsgBox "The ending date cannot be earlier than the starting date.",
vbOKOnly + vbExclamation, "Error"
        Exit Sub
    ElseIf cboEnYear.ListIndex = cboStYear.ListIndex And _
        cboEnMonth.ListIndex = cboStMonth.ListIndex And _
        cboEnDay.ListIndex <= cboStDay.ListIndex Then
        MsgBox "The ending date cannot be earlier than or equal to the
starting date.", vbOKOnly + vbExclamation, "Error"
        Exit Sub
    End If


    If Val(txtStoreDefAt) > 24 Or Val(txtStoreDefAt) < 0 Then
        MsgBox "The time of the day for the storing of the deflection
profiles is invalid.", vbOKOnly + vbExclamation, "Error"
        Exit Sub
    End If


'start the timer
STimer = Timer
SDate = Now


    NumOfRings = Val(frmMain.txtI)
    NumOfSectors = Val(frmMain.txtJ)
    NumOfSlices = Val(frmMain.txtK)


    gdColHeight = Val(frmMain.txtHeight)
    gdIRadius = Val(frmMain.txtIRadius)
    gdERadius = Val(frmMain.txtERadius)


    gdCompForce = Val(frmMain.txtF)
    gdEModulus = Val(frmMain.txtE)
    gdUnitLoad = Val(frmMain.txtUnitLoad)


    gdRefTemp = Val(frmMain.txtRefTemp)
    gdAThermal = Val(frmMain.txtAThermal)


        lstResults.Clear
        lstResults.AddItem ">> Main Data..."
        lstResults.AddItem "Column Height :" & myStr(gdColHeight) & " m."
        lstResults.AddItem "Internal Radius :" & myStr(gdIRadius) & " m."
        lstResults.AddItem "External Radius :" & myStr(gdERadius) & " m."


        lstResults.AddItem "Compression Force :" & myStr(gdCompForce) & "
MN."
```

```
        lstResults.AddItem "Elastic Modulus :" & myStr(gdEModulus) & " GPa."
        lstResults.AddItem "Unit Load :" & myStr(gdUnitLoad) & " KN/m³."
        lstResults.AddItem "Reference Temperature :" & myStr(gdRefTemp) & "
oC."
        lstResults.AddItem "Coefficient of thermal expansion :" &
myStr(gdAThermal) & " 1/oC."

        lstResults.AddItem "Number Of Rings :" & myStr(NumOfRings)
        lstResults.AddItem "Number Of Sectors :" & myStr(NumOfSectors)
        lstResults.AddItem "Number Of Slices :" & myStr(NumOfSlices)

        lstResults.AddItem ">> Calculating Geometry..."
    CalcGeometry

        lstResults.AddItem ">> Preliminary Calculations..."
    CalcInitialise

        lstResults.AddItem ">> Initialising..."
    CalcInitialiseDeltaE 'set the creep strains to zero
    CalcInitialiseDeflections 'erase previous solutions of deflections

        StDay = Val(cboStDay)
        StMonth = Val(cboStMonth)
        StYear = Val(cboStYear)
        EnDay = Val(cboEnDay)
        EnMonth = Val(cboEnMonth)
        EnYear = Val(cboEnYear)
        CurDay = StDay
        CurMonth = StMonth
        CurYear = StYear

        StoreDefEvery = Val(txtStoreDefEvery)
        StoreDefAt = Val(txtStoreDefAt)

        gdTimeInterval = Val(cboTimeInterval)
        glDaysElapsed = 0

myLoadDiagram 6, cboDiagram(6).ListIndex 'load the tstar diagram
myLoadDiagram 7, cboDiagram(7).ListIndex 'load the phi diagram

        lstResults.AddItem "Calculating from :" & FNDate(StDay, StMonth,
StYear) & " to " & FNDate(EnDay, EnMonth, EnYear) & " (dd/mm/yy)"
        lstResults.AddItem "Time Interval :" & Trim$(Str$(gdTimeInterval)) &
" hour(s)."
        If StoreDefEvery > 0 Then
            lstResults.AddItem "Storing deflection profiles every :" &
Trim$(Str$(StoreDefEvery)) & " days."
```

```
            lstResults.AddItem "Time of day (for the profiles) :" &
Trim$(Str$(StoreDefAt)) & " hour(s)."
        End If
        lstResults.Refresh

CancelHit = False
cmdCancel.Enabled = True

cmdCalculate.Enabled = False
cmdOk.Enabled = False
cmdDrawProfile.Enabled = False
cmdSaveReport.Enabled = False
For iA = 1 To NumOfDiagrams
    cboDiagram(iA).Enabled = False
    cmdDraw(iA).Enabled = False
Next iA
lstMonths.Enabled = False
cboTimeInterval.Enabled = False
cboStDay.Enabled = False
cboStMonth.Enabled = False
cboStYear.Enabled = False
cboEnDay.Enabled = False
cboEnMonth.Enabled = False
cboEnYear.Enabled = False
txtStoreDefEvery.Enabled = False
txtStoreDefAt.Enabled = False

Do While CurYear <= EnYear 'loop in years

    Do While CurMonth <= 12 'loop in months

        DaysOfCurMonth = FNDaysOfMonth(CurMonth, CurYear)

        'load the corresponding diagrams

            'lstResults.AddItem "Loading diagrams of " _
            & crMonthName(CurMonth) & Str$(CurYear)
        For iA = 1 To 5
            'lstResults.AddItem DiagramTitle(iA) & _
            " , combo listindex :" & Str$(CreepDiagramSel(CurMonth, iA))
            myLoadDiagram iA, CreepDiagramSel(CurMonth, iA)
        Next iA

        Do While CurDay <= DaysOfCurMonth 'in days
            glDaysElapsed = glDaysElapsed + 1
```

```
            DoEvents
            If CancelHit Then GoTo EXT

            lblInfo.Caption = "Calculating date : " & FNDate(CurDay,
CurMonth, CurYear)
            lblInfo.Refresh
                'calculate here
                    For dA = 0 To 24 - gdTimeInterval Step gdTimeInterval

                        gdTime = dA + gdTimeInterval / 2

                        'lstResults.AddItem "Calculating " & _
                        FNDate(CurDay, CurMonth, CurYear) & _
                        " , time :" & Str$(gdTime)

                        CalcTemperatures

                        CalcFreeStrains

                        CalcCurvatures

'    CalcVerifyEquations
'    If MsGCounter > 0 Then
'        lstResults.AddItem ">> Equation errors..."
'        For iC = 1 To MsGCounter
'            lstResults.AddItem MsG(iC)
'        Next iC
'    End If
                        CalcCreepStrains

                    Next dA

                If StoreDefEvery > 0 Then
                    If Fix(glDaysElapsed / StoreDefEvery) > NumOfStoredDef
Then
                        'store deflections here!
                        lblInfo.Caption = "Storing deflections : " &
FNDate(CurDay, CurMonth, CurYear)
                        lblInfo.Refresh

                        gdTime = StoreDefAt

                        CalcTemperatures

                        CalcFreeStrains
```

```
                        CalcCurvatures

                        CalcDeflections

                        CalcStoreDeflections FNDate(CurDay, CurMonth,
CurYear)

                        NumOfStoredDef = NumOfStoredDef + 1
                    End If
                End If

                'check if we have reached the ending date
                If CurDay = EnDay And CurMonth = EnMonth And CurYear =
EnYear Then GoTo Finish

            CurDay = CurDay + 1
        Loop

        CurDay = 1
        CurMonth = CurMonth + 1

    Loop 'in months

    CurMonth = 1
    CurYear = CurYear + 1
Loop 'in years

Finish:

lblInfo.Caption = ""

'store the final deflections profile
gdTime = StoreDefAt
CalcTemperatures
CalcFreeStrains
CalcCurvatures
CalcDeflections
CalcStoreDeflections FNDate(EnDay, EnMonth, EnYear)
NumOfStoredDef = NumOfStoredDef + 1

    lstResults.AddItem ">> Tip Deflection Calculations..."

For iC = 1 To DeflectionSCounter
    lstResults.AddItem "Date : " & DeflectionS(iC).DayCalculated
```

```
    lstResults.AddItem "X - Deflection :" & myStr(DeflectionS(iC).TipX) & "
m."
    lstResults.AddItem "Y - Deflection :" & myStr(DeflectionS(iC).TipY) & "
m."
    lstResults.AddItem "Z - Deflection :" & myStr(DeflectionS(iC).TipZ) & "
m."
Next iC

    cmdDrawProfile.Enabled = True
    cmdSaveReport.Enabled = True

    cmdCancel.Enabled = False

    cmdCalculate.Enabled = True
    cmdOk.Enabled = True
    For iA = 1 To NumOfDiagrams
        cboDiagram(iA).Enabled = True
        cmdDraw(iA).Enabled = True
    Next iA
    lstMonths.Enabled = True
    cboTimeInterval.Enabled = True
    cboStDay.Enabled = True
    cboStMonth.Enabled = True
    cboStYear.Enabled = True
    cboEnDay.Enabled = True
    cboEnMonth.Enabled = True
    cboEnYear.Enabled = True
    txtStoreDefEvery.Enabled = True
    txtStoreDefAt.Enabled = True

ETimer = Timer
EDate = Now

If Day(EDate) >= Day(SDate) Then 'assume we are in the same month
    SecS = 86400 * (Day(EDate) - Day(SDate)) + ETimer - STimer
Else 'assume 1 month has passed
    SecS = 86400 * (FNDaysOfMonth(Month(SDate), Year(SDate)) + Day(EDate) -
Day(SDate)) + ETimer - STimer
End If

iA = SecS \ 3600
If iA > 1 Then
    strTime = Str$(iA) & " hours"
ElseIf iA = 1 Then
    strTime = " 1 hour"
End If
```

```
SecS = SecS Mod 3600

iA = SecS \ 60
If iA > 1 Then
    strTime = strTime & Str$(iA) & " minutes"
ElseIf iA = 1 Then
    strTime = strTime & " 1 minute"
End If

iA = SecS Mod 60

If iA > 1 Then
    strTime = strTime & Str$(iA) & " seconds"
ElseIf iA = 1 Then
    strTime = strTime & " 1 second"
End If

    MsgBox "Calculation is completed." & Str$(glDaysElapsed) & _
    " days were calculated." & vbCrLf & _
    "Calculation time : " & strTime

Exit Sub

EXT:
cmdCancel.Enabled = False
lstResults.Clear
lblInfo.Caption = ""

cmdDrawProfile.Enabled = False
cmdSaveReport.Enabled = False

cmdCalculate.Enabled = True
cmdOk.Enabled = True
For iA = 1 To NumOfDiagrams
    cboDiagram(iA).Enabled = True
    cmdDraw(iA).Enabled = True
Next iA
lstMonths.Enabled = True
cboTimeInterval.Enabled = True
cboStDay.Enabled = True
cboStMonth.Enabled = True
cboStYear.Enabled = True
cboEnDay.Enabled = True
cboEnMonth.Enabled = True
```

```
cboEnYear.Enabled = True
txtStoreDefEvery.Enabled = True
txtStoreDefAt.Enabled = True

End Sub
```