Perturbation analysis (deterministic models)

These exercises require the latest version of `popdemo`. Head to `popdemo`'s GitHub (github.com/iainmstott/popdemo) for installation instructions (don't forget also to load the package using `library(popdemo)`!)

Complete the core exercises (in normal print) first, as code in each section continues from the last. Afterward, return to the "extras" sections (in *italics*), and try writing your own code. Code to be run is in chunks:

```
# a comment
an_input()
##   an output
```

Key terms in the text are in ***bold italic***, and functions or arguments are `fixed width`.

---

1. Data

We will use a matrix projection model (MPM) to explore population dynamics for the desert tortoise *Gopherus agassizzii*, with medium fecundity[1]. The population is found in the Mojave desert, USA. There are 8 stages are based on age and size (carapace length in mm):
- Yearling (age 0-1)
- Juvenile 1 (<60 mm)
- Juvenile 2 (90-99mm)
- Immature 1 (100-139mm)
- Immature 2 (140-179mm)
- Subadult (180-207mm)
- Adult 1 (208-239mm)
- Adult 2 (>240mm)

Load in the data:

```
data(Tort); Tort
##         Yr    J1    J2    I1    I2    SA    A1   A2
##   Yr 0.000 0.000 0.000 0.000 0.000 1.300 1.980 2.57
##   J1 0.716 0.567 0.000 0.000 0.000 0.000 0.000 0.00
##   J2 0.000 0.149 0.567 0.000 0.000 0.000 0.000 0.00
##   I1 0.000 0.000 0.149 0.604 0.000 0.000 0.000 0.00
##   I2 0.000 0.000 0.000 0.235 0.560 0.000 0.000 0.00
##   SA 0.000 0.000 0.000 0.000 0.225 0.678 0.000 0.00
##   A1 0.000 0.000 0.000 0.000 0.000 0.249 0.851 0.00
##   A2 0.000 0.000 0.000 0.000 0.000 0.000 0.016 0.86
```

The numbers in the matrix (called ***matrix elements*** or ***transitions***) describe the probability of moving FROM stages in each column TO stages in each row, within the time interval chosen. For example, for this desert tortoise matrix, in any year a subadult (stage 6) has approimately 24.9% probability of becoming an adult (stage 7): this may be called a growth or progression transition. Likewise, in any year a subadult has about 67.8% chance of staying a subadult: this is called a stasis transition. This means that 100 - (67.8 + 24.9) = 7.3% of subadults die every year. There are different types of transitions: in this matrix there are also fecundity transitions which describe offspring production, and subadults produce on average 1.3 offspring per year. Other species may have different transitions, including skipping stages through fast growth, shrinkage or fission (especially in modular organisms, e.g. most plants, corals), or asexual reproduction.

Matrix elements combine underlying ***vital rates*** such as survival, growth and reproduction. For example, the subadult to adult transition (24.9%) combines probability of growing to adult size in one year, and probability of surviving the year. The subadult fecundity (1.3) combines the average number of offspring produced by subadults per year, and probability that those offspring survive the year.

[1] Doak et al. (1994) Ecol. Appl., 4, 446-460.

Figure 1: A baby desert tortoise

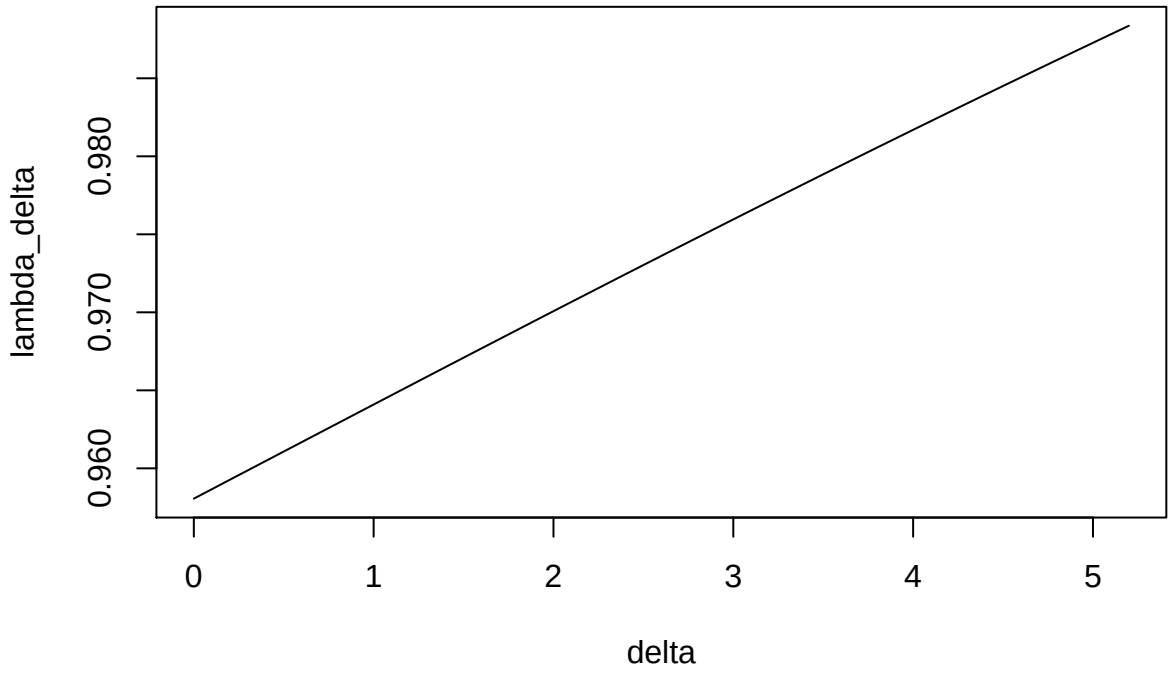
---

2. Perturbation analysis fundamentals

This vignette builds on the "Deterministic population dynamics" vignette, in which the population projections, asymptotic dynamics and transient dynamics are introduced. These dynamics depend on the life cycle transitions described in the projection matrix elements. If the matrix elements change, then the population dynamics change too. However, different life cycle transitions contribute differently to population dynamics: for example, increasing the survival of younger age classes may cause a smaller increase in asymptotic growth than increasing the survival of older age classes. ***Perturbation analysis*** looks to understand and compare how such changes may impact population dynamics: this information can help inform population management by identifying the best management strategies to achieve desired results.

It's easy to see this by changing a matrix element (e.g. fertility of subadults), by some magnitude of perturbation (δ), calculating a population dynamic (e.g. asymptotic growth), and plotting the results:

```
delta <- seq(0, 4*Tort[1, 6], 0.1)
Tort_delta <- Tort
lambda_delta <- numeric(length(delta))
for(i in 1:length(delta)){
    Tort_delta[1, 6] <- Tort[1, 6] + delta[i]
    lambda_delta[i] <- eigs(Tort_delta, "lambda")
}

plot(delta, lambda_delta, type = "l")
```



---

3. Perturbation analysis of asymptotic growth

Perturbation analyses of asymptotic growth can often be expressed in relatively simple ways.

3.1 Sensitivity and elasticity: linear perturbation analysis

The first perturbation analyses described linear relationships between matrix elements and asymptotic growth. These are the first derivatives of the relationship between a the matrix element and lambda:

$$s_{i,j} = \frac{\partial \lambda_{max}}{\partial a_{i,j}} = \frac{v_i w_j}{\mathbf{v}^{\mathrm{T}}\mathbf{w}}$$

$\mathbf{v}$ and $\mathbf{w}$ are commonly scaled so that $||\mathbf{w}||_1 = 1$ and $\mathbf{v}^{\mathrm{T}}\mathbf{w} = 1$ (see "Deterministic population dynamics vignette), so the above simplifies to $v_i w_j$. This equation describes the slope of the line in the previous plot. It's evaluated for infinitessimally small perturbation magnitude, so strictly it's the slope of the line where delta = 0.

3.1.1 Sensitivity

The above analysis is called ***sensitivity analysis*** and it's quick to calculate sensitivity of all matrix elements using:

$$\mathbf{S} = \frac{\mathbf{v}\mathbf{w}^{\mathrm{T}}}{\mathbf{v}^{\mathrm{T}}\mathbf{w}}$$

This is the "sensitivity matrix", and assuming the same scaling of $\mathbf{w}$ and $\mathbf{v}$, simplifies to $\mathbf{v}\mathbf{w}^{\mathrm{T}}$. It can be calculated in `popdemo` using the `sens` function.

```
sens(Tort)
##           [,1]   [,2]   [,3]   [,4]   [,5]     [,6]    [,7]    [,8]
##   [1,] 0.00000 0.0000 0.0000 0.0000 0.0000 0.006034 0.01403 0.00229
##   [2,] 0.05798 0.1062 0.0000 0.0000 0.0000 0.000000 0.00000 0.00000
##   [3,] 0.00000 0.2786 0.1062 0.0000 0.0000 0.000000 0.00000 0.00000
##   [4,] 0.00000 0.0000 0.2786 0.1173 0.0000 0.000000 0.00000 0.00000
##   [5,] 0.00000 0.0000 0.0000 0.1767 0.1043 0.000000 0.00000 0.00000
##   [6,] 0.00000 0.0000 0.0000 0.0000 0.1845 0.148243 0.00000 0.00000
##   [7,] 0.00000 0.0000 0.0000 0.0000 0.0000 0.135231 0.31452 0.00000
##   [8,] 0.00000 0.0000 0.0000 0.0000 0.0000 0.000000 0.36781 0.06001
```

Any element equal to zero in the projection matrix is shown as zero in the sensitivity matrix, but adding `all = TRUE` will show the sensitivity of every element.

3.1.2 Elasticity

The effort required to change a matrix element by a certain amount is not the same for every matrix element. The fecundity of subadults is 1.3 but the fecundity of the second adult class is 2.57. Increasing fecundity by 1 may therefore be harder for subadults than adults. Elasticity analysis adjusts for these differences:

$$\mathbf{E} = \frac{1}{\lambda_{max}} \mathbf{A} \cdot \mathbf{S}$$

It's calculated using the `elas` function:

```
elas(Tort)
##           Yr      J1      J2      I1      I2       SA       A1       A2
##   Yr 0.00000 0.00000 0.00000 0.00000 0.00000 0.008188 0.029004 0.006143
##   J1 0.04333 0.06283 0.00000 0.00000 0.00000 0.000000 0.000000 0.000000
##   J2 0.00000 0.04333 0.06283 0.00000 0.00000 0.000000 0.000000 0.000000
##   I1 0.00000 0.00000 0.04333 0.07393 0.00000 0.000000 0.000000 0.000000
##   I2 0.00000 0.00000 0.00000 0.04333 0.06096 0.000000 0.000000 0.000000
##   SA 0.00000 0.00000 0.00000 0.00000 0.04333 0.104908 0.000000 0.000000
##   A1 0.00000 0.00000 0.00000 0.00000 0.00000 0.035147 0.279375 0.000000
##   A2 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000 0.006143 0.053872
```

3.2 Transfer functions: nonlinear perturbation analysis

In actual fact, the sensitivites asymptotic growth (or transient dynamics), are linear approximations of nonlinear relationships. The effect of changing a life cycle transition or a vital rate on population dynamics can be very nonlinear: sensitivities are tangents to these functions at infinitessimally small perturbation magnitudes of near 0 (differentiations of a curve).

`popdemo` provides tools for describing and visualising the nonlinear relationships between matrix entries and population dynamics, called ***transfer functions***. These functions paradoxically define the perturbation magnitude in terms of λ_{max}. The perturbed matrix can be represented using:

$$\mathbf{A}_\delta = \mathbf{A} + \delta \mathbf{d}\mathbf{e}^{\mathrm{T}}$$

Where $\mathbf{d}$ represents the row to be perturbed, $\mathbf{e}$ represents the column to be perturbed, and δ is the perturbation magnitude. Using this notation, it can be shown[2] that:

$$\delta^{-1} = \mathbf{e}^{\mathrm{T}}(\lambda \mathbf{I} - \mathbf{A})^{-1}\mathbf{d}$$

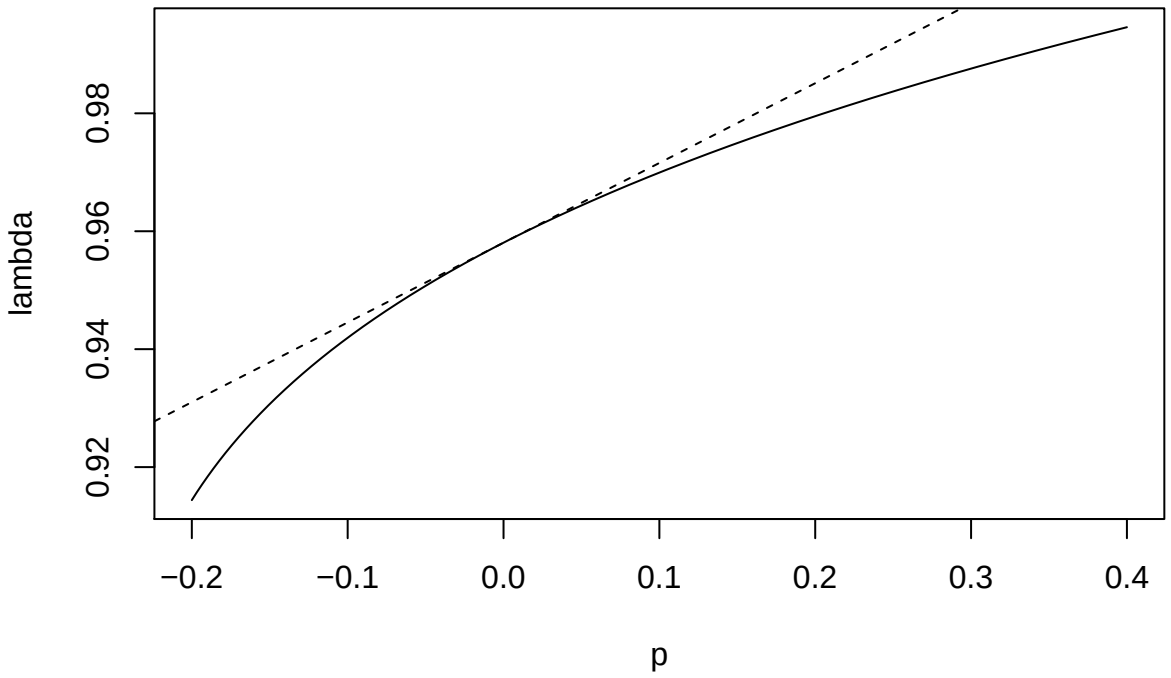
[2] Hodgson & Townley (2004) J. Appl. Ecol., 41, 1155-1161.

Where λ is the asymptotic growth of $\mathbf{A}_\delta$. Using these equations, it is possible to: 1. Decide on a maximum and minimum perturbation value 2. Compute λ of $\mathbf{A}_\delta$ for max and min δ 3. Compute the transfer function for values of λ within this range

Alternatively, if there is a desired target range for growth, stages 1 and 2 can be skipped. It's important to make sure that the range of δ doesn't result in any negative matrix elements, or any situation where overall survival of a single stage is greater than 1.

The `tfa_lambda` function calculates this transfer function ('tfa' stands for 'transfer function analysis'). The following calculates perturbation to element [7,6]. The sensitivity is the slope where delta is equal to zero, which has an intercept at λ_{max}:

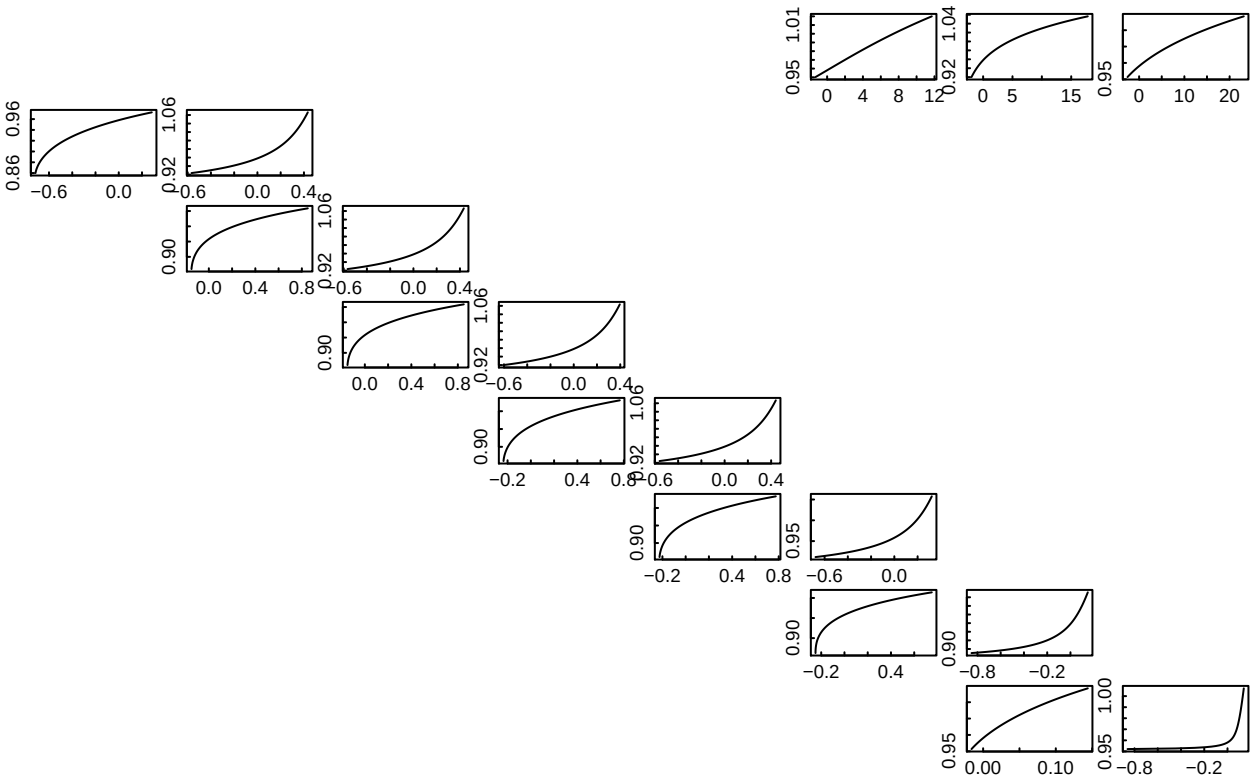
```
delta <- seq(-0.2, 0.4, 0.01)
d1 <- c(0, 0, 0, 0, 0, 0, 1, 0)
e1 <- c(0, 0, 0, 0, 0, 1, 0, 0)
tf1 <- tfa_lambda(Tort, d = d1, e = e1, prange = delta)

plot(tf1)
s76 <- sens(Tort)[7, 6]
abline(eigs(Tort, "lambda"), s76, lty = 2)
```



This transfer function shows that the perturbation has a nonlinear effect on λ. Note that it's also possible to specify a target range of λ instead using the `lambdarange` argument.

The function `tfam_lambda` calculates a transfer function for every matrix element (the 'm' stands for 'matrix'), and it's possible to plot these together:

```
tfml <- tfam_lambda(Tort)
plot(tfml)
```



iii. EXTRAS...
Transfer functions don't have to be limited to single matrix elements. By choosing the **d** *and* **e** *vectors carefully, it's possible to calculate more complex management strategies. For example, by choosing* `d <- c(1, 0, 0, 0, 0, 0, 0, 0)` *and* `e <- c(0, 0, 0, 0, 0, 1, 1, 1)`*, the transfer function perturbs all fecundity values simultaneously. Trade-offs in management can also be modelled: using* `d <- c(0, 0, 0, 0, 0, -1, 1, 0)` *and* `e <- c(0, 0, 0, 0, 0, 1, 0, 0)` *would model increases in the rate of growth of subadults to adults whilst simultaneously decreasing stasis of subadults, so keeping overall survival the same. Try using these perturbation structures, or exploring other possibilities (not all combinations of elements are possible, as outlined in Hodgson & Townley 2004).*

---

4. Perturbation analysis of population inertia

Population inertia is an index that is very amenable to mathematical trickery. Although it is a measure of transient dynamics (transient dynamics lead to the population having an inertia), it is calculated using the dominant eigenvectors of the matrix. This makes it very amenable to perturbation analysis, unlike other transient indices, as the eigenvectors of $\mathbf{A}_\delta$ can be expressed in terms of $\mathbf{A}$, λ, $\mathbf{d}$ and $\mathbf{e}$. The equations get long, but can be found in Stott et al. (2012)[3].

4.1 Sensitivity: linear perturbation analysis

Calculating the sensitivity values for inertia is super simple. Let's do it for a specific population vector first[4]:

[3] Stott et al. (2012) Methods Ecol. Evol., 3, 673-684.
[4] The `tolerance` argument sometimes has to be increased, when there are problems computing the sensitivities.

```
Tortvec <- c(1, 1, 2, 3, 5, 8, 13, 21)
tfsm_inertia(Tort, Tortvec, tolerance=1e-5)
##         [,1]   [,2]   [,3]   [,4]   [,5]    [,6]    [,7]    [,8]
##   [1,] 0.000  0.000  0.000  0.000  0.000  0.1059  0.1428  0.7018
##   [2,] 1.828  2.235  0.000  0.000  0.000  0.0000  0.0000  0.0000
##   [3,] 0.000 -2.469 -2.001  0.000  0.000  0.0000  0.0000  0.0000
##   [4,] 0.000  0.000 -6.600 -3.867  0.000  0.0000  0.0000  0.0000
##   [5,] 0.000  0.000  0.000 -4.906 -3.351  0.0000  0.0000  0.0000
##   [6,] 0.000  0.000  0.000  0.000 -4.715 -3.4406  0.0000  0.0000
##   [7,] 0.000  0.000  0.000  0.000  0.000 -2.4694 -8.0662  0.0000
##   [8,] 0.000  0.000  0.000  0.000  0.000  0.0000 -8.7689 16.3527
```

Many of the matrix transitions have a negative effect on population inertia, but there's a large positive effect of the survival of the largest adults.

We can also calculate sensitivities for the bounds on population dynamics.

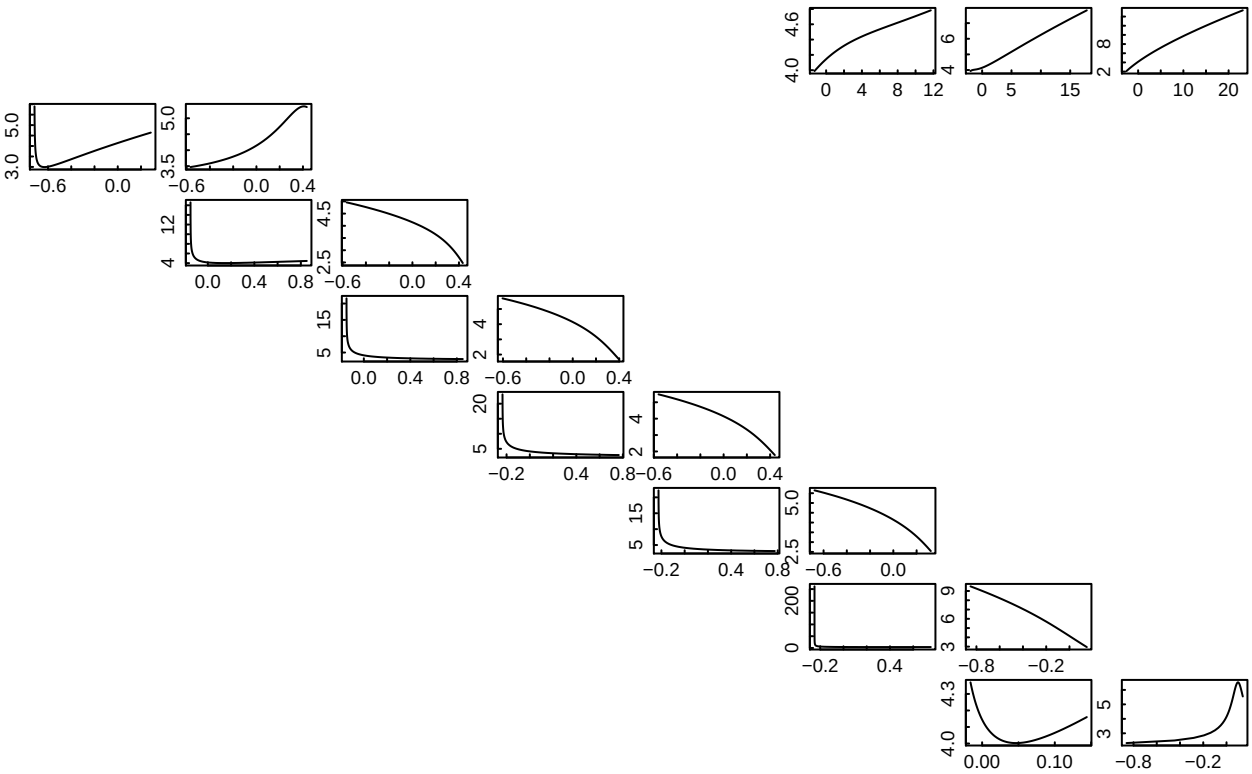
```
tfsm_inertia(Tort, bound="upper", tolerance=1e-5)
##         [,1]   [,2]   [,3]   [,4]   [,5]     [,6]     [,7]   [,8]
##   [1,] 0.000  0.000  0.000  0.000  0.000 -0.02837  -0.7376  1.754
##   [2,] 2.617  3.401  0.000  0.000  0.000  0.00000   0.0000  0.000
##   [3,] 0.000 -1.380 -1.917  0.000  0.000  0.00000   0.0000  0.000
##   [4,] 0.000  0.000 -6.697 -4.515  0.000  0.00000   0.0000  0.000
##   [5,] 0.000  0.000  0.000 -5.665 -4.687  0.00000   0.0000  0.000
##   [6,] 0.000  0.000  0.000  0.000 -6.792 -8.16874   0.0000  0.000
##   [7,] 0.000  0.000  0.000  0.000  0.000 -6.62430 -30.4595  0.000
##   [8,] 0.000  0.000  0.000  0.000  0.000  0.00000 -34.7996 43.437

tfsm_inertia(Tort, bound="lower", tolerance=1e-5)
##          [,1]   [,2]   [,3]    [,4]    [,5]     [,6]       [,7]      [,8]
##   [1,] 0.0000 0.0000 0.0000 0.00000 0.00000  0.01094 -0.0001746 -0.004586
##   [2,] 0.2155 0.3414 0.0000 0.00000 0.00000  0.00000  0.0000000  0.000000
##   [3,] 0.0000 0.5028 0.1385 0.00000 0.00000  0.00000  0.0000000  0.000000
##   [4,] 0.0000 0.0000 0.3000 0.06148 0.00000  0.00000  0.0000000  0.000000
##   [5,] 0.0000 0.0000 0.0000 0.13605 0.02909  0.00000  0.0000000  0.000000
##   [6,] 0.0000 0.0000 0.0000 0.00000 0.10871 -0.01614  0.0000000  0.000000
##   [7,] 0.0000 0.0000 0.0000 0.00000 0.00000  0.01684 -0.5351484  0.000000
##   [8,] 0.0000 0.0000 0.0000 0.00000 0.00000  0.00000 -0.5944875 -0.216656
```

Note that the sensitivities of the different bounds are different. This will be the same for any population vector: the sensitivities depend on the population structure.

4.2 Transfer functions: nonlinear perturbation analysis

`popdemo` provides tools for visualising the nonlinear relationships between matrix entries and population inertia, which as for asymptotic growth can be calculated using transfer functions. Transfer function analysis for every matrix element can be achieved using `tfam_inertia`:

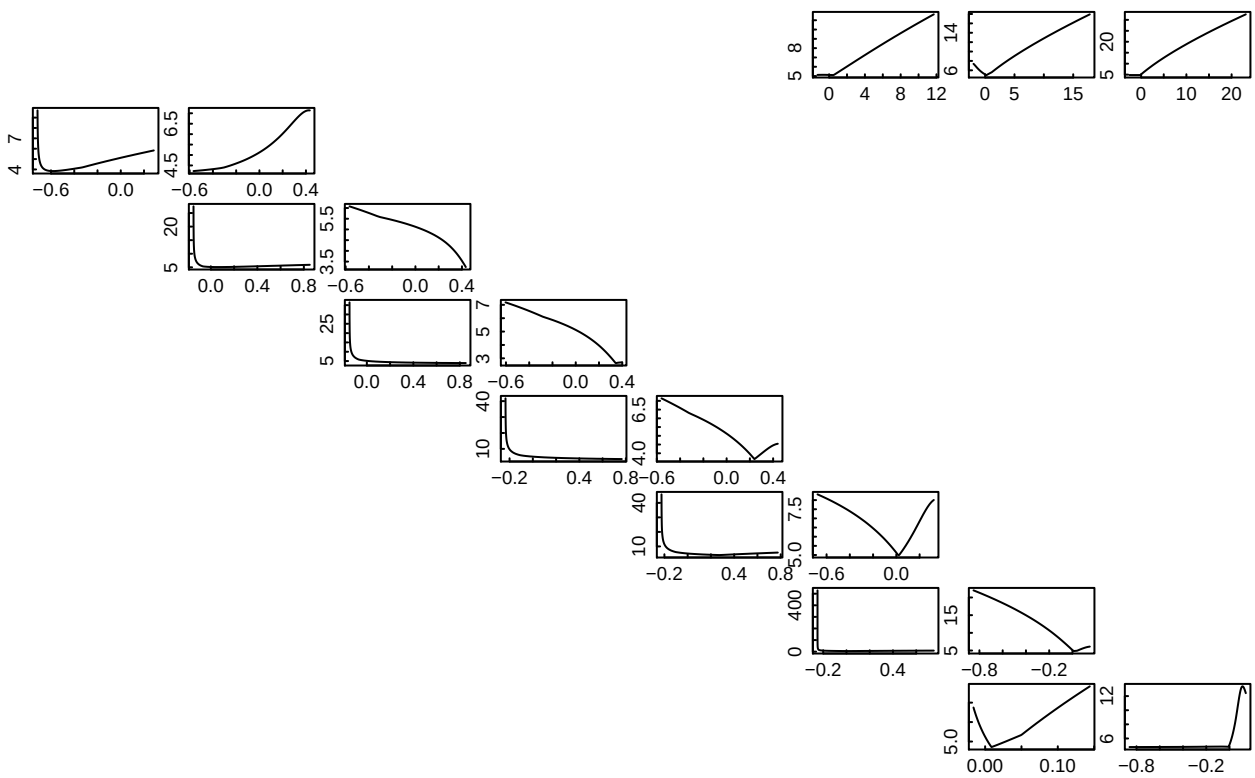
```
tfmi <- tfam_inertia(Tort, vector = Tortvec)
plot(tfmi)
```



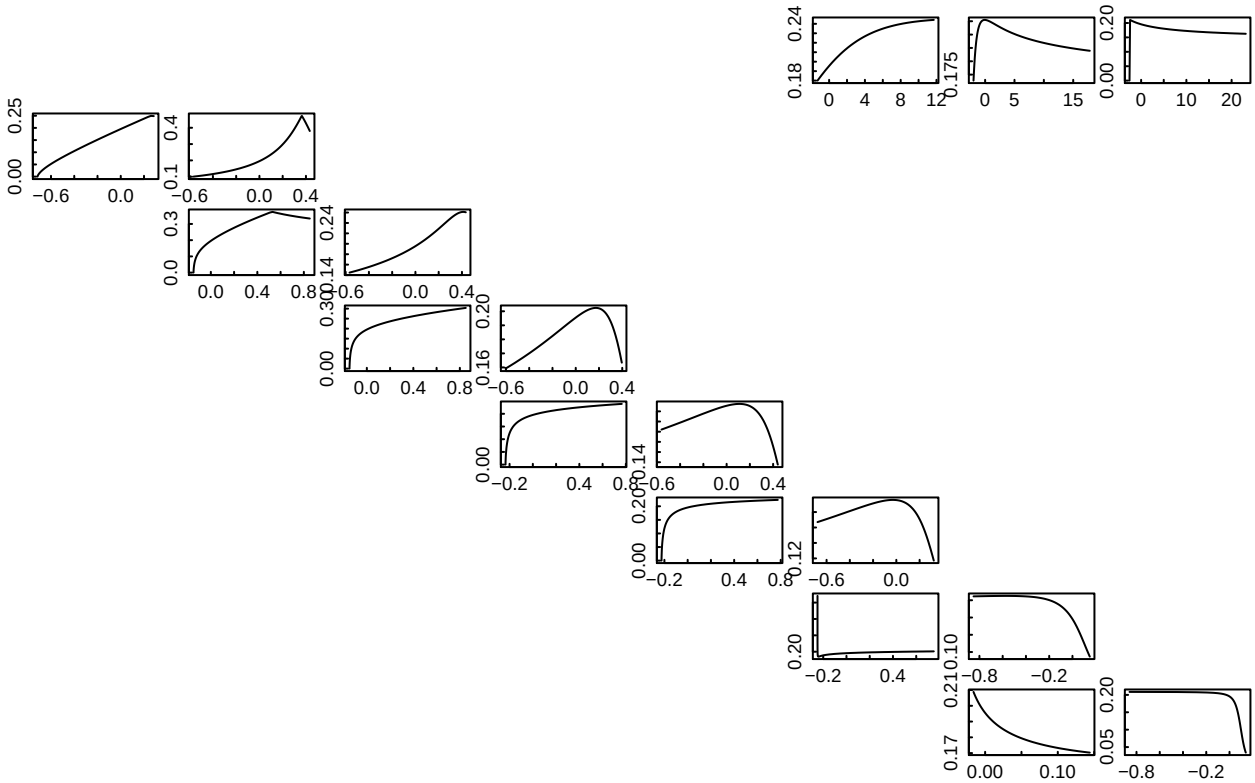
On the x axes are the perturbation magnitudes (how much you change the entry of the matrix by), and on the y axis is the value of inertia. Note how nonlinear the curves can be, and that unlike for asymptotic growth the slopes can be negative. The sensitivities are tangents to these curves at p=0.

We can do the same thing for the bounds:

```
tfmi_upr <- tfam_inertia(Tort, bound="upper")
plot(tfmi_upr)
```




```
tfmi_lwr<-tfam_inertia(Tort, bound="lower")
plot(tfmi_lwr)
```



These can be incredibly nonlinear, because there can be breaks in the function where a different stage-biased population structure takes over as achieving the bound on inertia. Note where there are kinks in the lines, or switches in the direction of the slope.

iv. EXTRAS...

In the same way that inertia depends strongly on the population vector, perturbation analysis of inertia does as well: try passing different numbers to `vector`.

In the `tfsm_inertia` *function, 'tfsm' stands for 'transfer function sensitivity matrix'. This is because the sensitivity is calculated by differentiating the transfer function. There is also a* `tfs_inertia` *function, which calculates the sensitivity of a particular transfer function specified with* **d** *and* **e** *vectors in the same way as the* `tfa_lambda` *and* `tfa_inertia` *functions.*

Worked real-world examples for transfer functions of inertia and lambda can be found in Stott et al. (2012)