Generalised Shape Constraints (Draft)

Charlotte Maia

October 16, 2011

Abstract

This vignette provides an overview of the gsc package for representing generalised shape constraints and transforming regularly spaced series to satisfy those constraints. The author regards generalised shape constraints as sets of constraints on the signs of the zeroth, first, second and third derivatives. Quadratic programming is used to implement such transformations, which (unfortunately) tend to produce angular curves. Currently, the objective function incorporates a roughness penalty to reduce this problem. However, this approach requires a roughness parameter, which creates further problems.

1 Introduction

This package provides a function for transforming a regularly spaced series, such that the transformed series satisfies a set of shape constraints. For short regularly spaced series, the transformation can be used as a stand alone smoothing technique. For long or irregularly spaced series, the data needs to smoothed first, via another method, then the transformation can be applied to the intermediate series.

The author uses the term generalised shape constraints, to describe shape constraints that accommodate an almost arbitrary combination of constraints of the signs of a smooth function's values and it's first few derivatives or a regularly spaced series approximating that function. Currently, constraints can be applied to the zeroth (the function itself), first, second or third derivatives. A common example is a convex:increasing function which has positive first and second derivatives.

Piece-wise constraints are supported and we could fit a convex-concave function, with a knot specifying where the convexity changes from positive to negative. It's worth noting that some convex-concave curves can be reformulated by constraining the third derivative. However, there are many exceptions to this.

The transformation is accomplished via quadratic programming and this package makes use of the quadprog package. The formulation as a quadratic program is partially described in an appendix.

Unfortunately, transformations can produce angular curves.

The current version of this package, includes a roughness penalty. The penalty is formulated as the sum of the squares of the transformed series, after differencing it a few times, to approximate a higher order derivative.

The roughness penalty reduces the angularity problem, however it requires a roughness parameter, which is awkward to determine.

Currently, if the parameter is too high, the transformation fails, presumably due to an absence of an optimal solution. To avoid this problem, the examples used in this vignette use a default parameter, which is assumed to be safe. The examples could be improved by experimenting with the parameter.

Note that this package should be regarded as unstable and experimental, this package's functions may be changed in the future.

In addition to the angularity issues, there are some minor issues with piecewise-wise constraints, especially with constant segments.

2 Generalised Shape Constraints

A generalised shape constraint (gsc) object is created by calling the gsc function. A trivial gsc object, specifying no constraints can be created as follows:

```
> gsc ()
```

```
gsc object
----------
s0: none
s1: none
s2: none
s3: none
knots: none
```

Shape constraints for a particular derivative are specified using a single string, consisting of the symbols "0" for constant, "+" for positive (strictly speaking, non-decreasing), "-" for negative (strictly speaking, non-increasing) and "?" for unconstrained. For a global constraint (where there are no knots), the string is a single symbol. For piece-wise constraints, in principle, the number of constraint symbols should match the number of segments (the number of knots plus one), however a single symbol is replicated.

So "0+-?" refers to "constant-positive-negative-unconstrained".

The first four arguments of the gsc function describe the zeroth, first, second and third derivatives. Hence to create a gsc object representing a positive first derivative (increasing), we write:

```
> #increasing
> gsc (,"+")
```

```
gsc object
----------
s0: none
s1: +
s2: none
s3: none
knots: none
```

For a positive second derivative (convex):

```
> #convex
> gsc (,,"+")
```

```
gsc object
----------
s0: none
s1: none
s2: +
s3: none
knots: none
```

For an convex:increasing constraint:

```
> #convex:increasing
> gsc (,"+", "+")
```

```
gsc object
----------
s0: none
s1: +
s2: +
s3: none
knots: none
```

For piecewise constraints we need to specify internal knots. So an increasing-constant-increasing constraint with knots at 10 and 20 would be:

```
> #increasing-constant-increasing
> gsc (,"+0+", knots=c (10, 20) )
```

```
gsc object
----------
s0: none
s1: +0+
s2: none
s3: none
knots: 10, 20
```

Note that, in general, the knots need to be elements of the x values of the series we wish to transform. More on this later.

Where the constraint on one derivative is intended to be global and another is intended to be piecewise we can take advantage of replication. So for a constraint that's globally positive (in it's zeroth derivative) and piecewise straight-convex-concave with knots at 50 and 200:

```
> #straight-convex-concave:positive
> gsc ("+",,"0+-", knots=c (50, 200) )
```

```
gsc object
----------
s0: +++
s1: none
s2: 0+-
s3: none
knots: 50, 200
```

3 Transformation Overview

The function gsc_solve is used to transform a series, re-iterating that the series needs to be regularly spaced. The current version of the function, takes three main arguments:

```
> args (gsc_solve)

function (m, x, y, order = 4, reweight = TRUE, p = 0.98)
NULL
```

Where m is a gsc object, x and y represent an unconstrained series and the other arguments control the way the roughness penalty is computed. The function returns a vector (which I denote v), representing the transformed values of y.

Increasing the order (two to six) or increasing the the roughness parameter p ($0 \leq p < 1$) increases the smoothness. However, if the values are too high, then an error is produced, which I need to explore further.

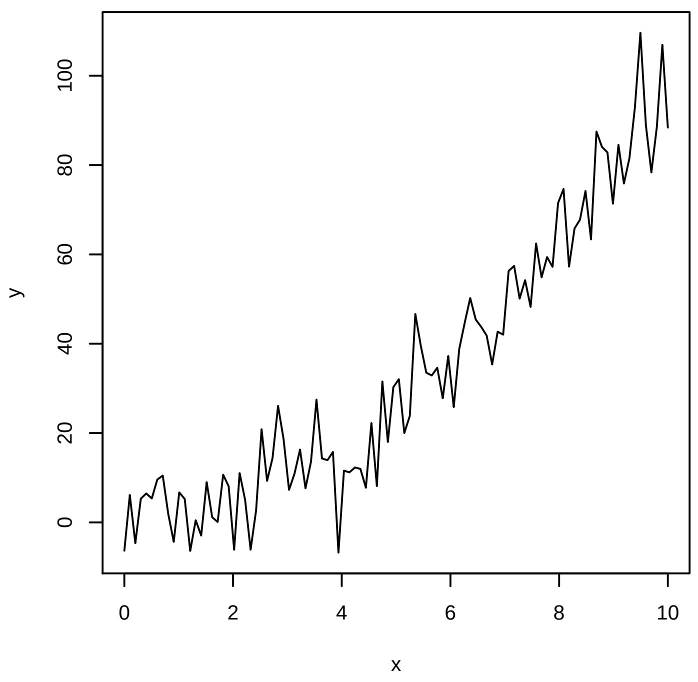
The values of x don't effect the solution, except where piece-wise constraints are used.

4 Convex:Increasing Example

Some simulated data (approximately quadratic):

```
> n = 100
> x = seq (0, 10, length=n)
> y = x^2 + 8 * rnorm (n)

> plot (x, y, type="l")
```



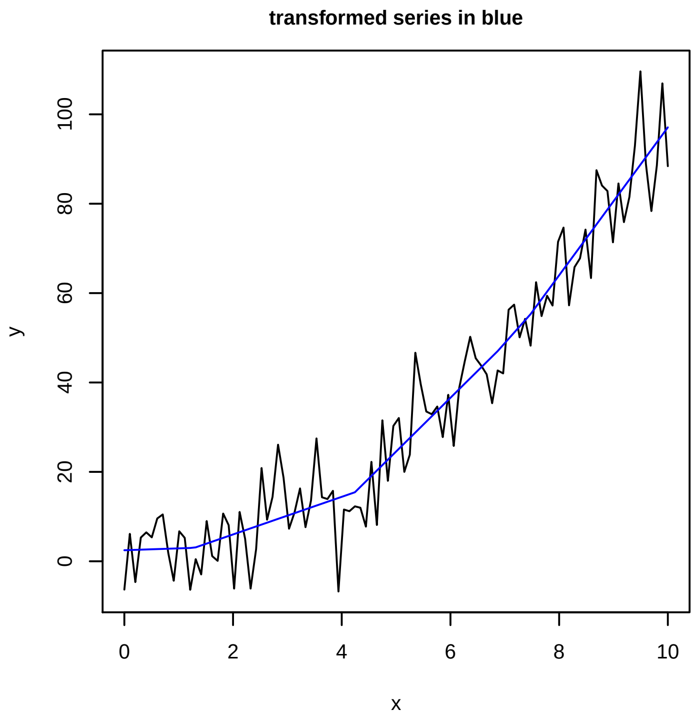
A gsc object (convex-increasing):

```
> m = gsc (,"+", "+")
```

Compute the transformed series:

```
> v = gsc_solve (m, x, y)
```

```
> plot (x, y, type="l", main="transformed series in blue")
> lines (x, v, col="blue")
```



5 Third Derivative Example

We may be able to use a constraint on the third derivative to transform the inverse of an empirical CDF. Actually this approach is incorrect, however I didn't realise it until I finished writing the vignette, so I decided to leave in...

Giving a random sample, sorted with no duplicates:

```
> n_raw = 80
> x = unique (rnorm (n_raw) )
> n = length (x)
> x = sort (x)
```

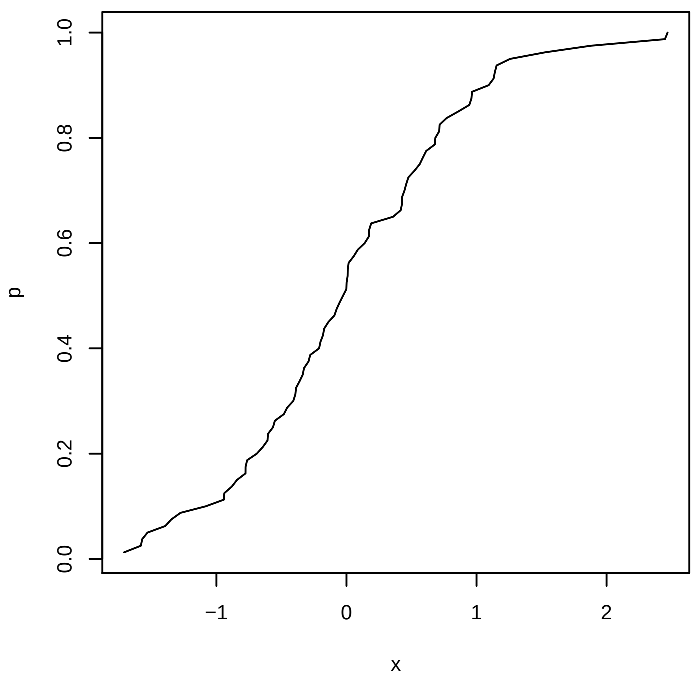
We can compute cumulative probabilities, corresponding to it's quantiles:

```
> p = (1:n) / n
```

Note that whilst the above calculation for p is suitable for step functions, it's unsuitable for continuous functions, however, I'll use it anyway, for simplicity.

The series x and p (which is irregular in x) gives us the ECDF:

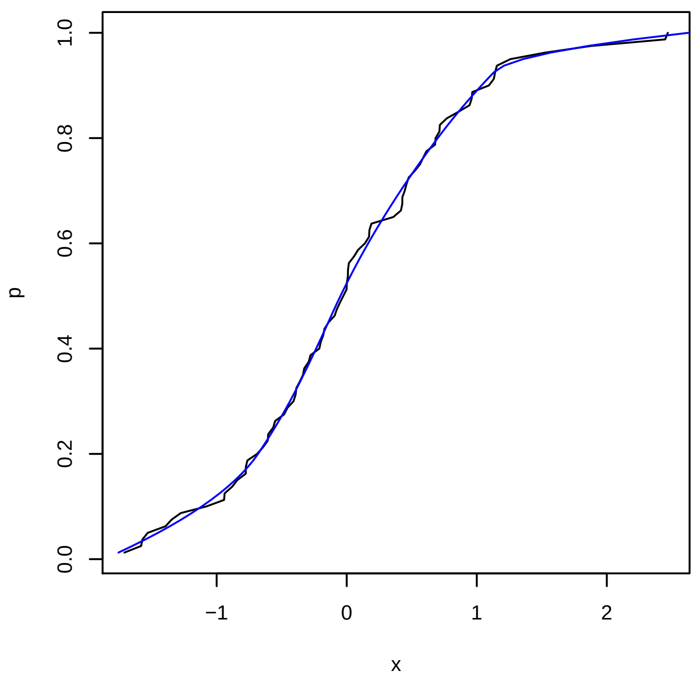
```
> plot (x, p, type="l")
```



Whilst it's irregular in x, it's regular in p, hence we can apply a constraint to the third (and first) derivative, using the transposed series:

```
> m = gsc (,"+",,"+")
> v = gsc_solve (m, p, x)

> plot (x, p, type="l")
> lines (v, p, col="blue")
```



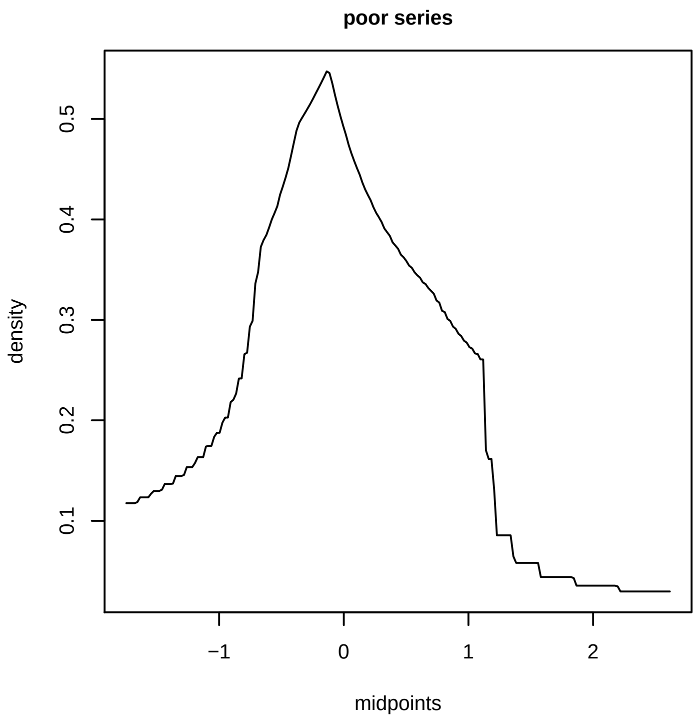
Note that contrary to other sections of this vignette, v is the transformed x values rather than the transformed y values.

At first glance, it may seem like a reasonable curve, however some very rough numerical differentiation, exposes it's problems.

```
> #resample first, series x_ and p_
> n_ = 200
> x_ = seq (min (v), max (v), length=n_)
> p_ = c (1 / n, numeric (n_ - 2), 1)
> for (i in 2:(n_ - 1) )
 {        k = sum (x_ [i] >= v)
         p_ [i] = k / n + (x_ [i] - v [k]) / (v [k + 1] - v [k]) / n
 }

> #differentiate
> dx = (x_ [n_] - x_ [1]) / (n_ - 1)
> dp = diff (p_)
> midpoints = x_ [-n_] + dx / 2
> density = dp / dx

> plot (midpoints, density, type="l", main="poor series")
```

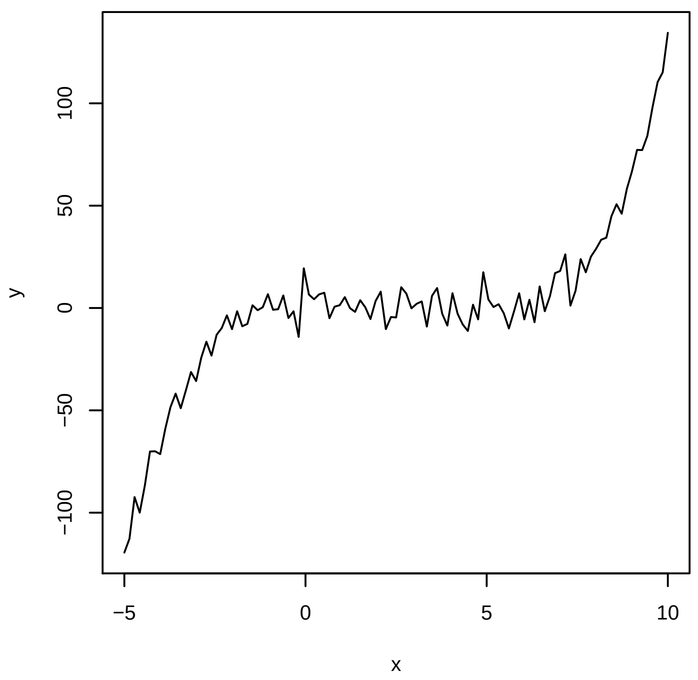


6 Concave-Straight-Convex Example

More simulated data.

```
> n = 107
> x = seq (-5, 10, length=n)
> y = rep (0, n)
> y [x < 0] = x [x < 0]^3
> y [x > 5] = (x [x > 5] - 5)^3
> y = y + 6 * rnorm (n)
```

```
> plot (x, y, type="l")
```



In general, the knots we use need to be elements of x. If the knots aren't elements of x then each knot defines an interval between two points, where there are no constraints.

Intuitively, there are knots at zero and five, however these need to be adjusted to satisfy the criteria above.

```
> closest = function (x, k)
 {        dist = abs (x - k)
         i = which.min (dist)
         x [i]
 }
> k1 = closest (x, 0)
> k2 = closest (x, 5)
```

We can regard the series as a concave-straight-convex curve:

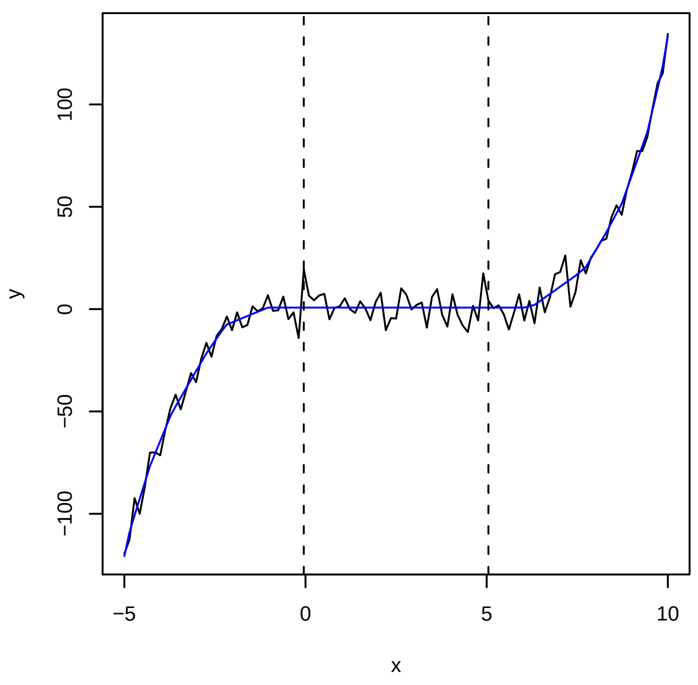
```
> m = gsc (,"+++", "-0+", knots=c (k1, k2) )
> m
```

```
gsc object
----------
s0: none
s1: +++
s2: -0+
s3: none
knots: -0.0471698113207548, 5.04716981132075
```

Note that we could constrain the second segment's first derivative to produce a constant segment, however the series tends to be slightly jagged. If we do this we need to remove the second segment's second derivative constraint of straightness. I've added further increasing constraints.

```
> v = gsc_solve (m, x, y)
```

```
> plot (x, y, type="l")
> lines (x, v, col="blue")
> abline (v=c (k1, k2), lty=2)
```



Appendix: Quadratic Program

The package minimises the following function:

$$(1-p)\text{expr}_1 + (p)\text{expr}_2$$

For the second order case, the the subexpressions above expand as follows:

$$\text{expr}_1 \rightarrow \sum_{\forall i} (y_i - v_i)^2$$

$$\text{expr}_2 \rightarrow \sum_{\forall i \in 1:(n-2)} (v_{[i+2]} - 2v_{[i+1]} + v_{[i]})^2$$

Where y is the (constant) untransformed series, v is the (unknown) transformed series (to be solved for) and the expression $\forall i \in 1 : (n-2)$ implies that we iterate over all indices, except the last two.

Noting that p is the weight of the roughness penalty and the second subexpression is the unweighted roughness penalty.

For third order case the second subexpression is replaced by:

$$\text{expr}_2 \rightarrow \sum_{\forall i \in 1:(n-3)} (v_{[i+3]} - 3v_{[i+2]} + 3v_{[i+1]} - v_{[i]})^2$$

For the fourth:

$$\text{expr}_2 \rightarrow \sum_{\forall i \in 1:(n-4)} (v_{[i+4]} - 4v_{[i+3]} + 6v_{[i+2]} - 4v_{[i+1]} + v_{[i]})^2$$

Formulation of the constraints is more complex.

The previous version of this package didn't use a roughness penalty and I might return to that approach. Future versions may incorporate smoothness (or roughness) into the constraints or formulate the roughness penalty differently.