# Tensors for matrix differentiation

Richard Turner

Here are some notes on how to use tensors to find matrix derivatives, and the relation to the .∗ (Hadamard), vec, ⊗ (Kronecker), vec-transpose and reshape operators. I wrote these notes for myself, and I apologise for any mistakes and confusions. *Two sections are currently unfinished: I hope to complete them soon.*

## 1 A tensor notation

Let's setup one useful form of tensor notation, which incorporates the matrix and inner product, the outer product, the Hadamard (`MATLAB` .* or $\circ$) product diag and $\text{diag}^{-1}$. These will be denoted using different combinations of pairs of up-stairs and down-stairs indices. If we have only $2^{nd}$ order tensors (and lower) we want to be able to easily convert the result into matrix representation. We have a free choice for the horizontal ordering of indices therefore this can be used to denote transposes and the order of multiplication.

$$
\begin{aligned}
a_i{}^j b_j{}^k &= \sum_j a_i{}^j b_j{}^k && (1)\\
&= (AB)_i{}^k && (2)\\
A_i{}^j &= (A^T)^j{}_i && (3)\\
a_i b^j &= (\mathbf{a}\mathbf{b}^T)_i{}^j && (4)\\
a_i{}^i &= \sum_i a_i{}^i && (5)\\
&= \text{tr}\, A && (6)\\
A_i{}^j B_i{}^j &= H_{iln}^{jkm} A_k{}^l B_m{}^n && (7)\\
&= (A \circ B)_i{}^j && (8)\\
H_{iln}^{jkm} &= \delta_i{}^k \delta_l{}^j \delta_i{}^m \delta_n{}^j && (9)\\
A_{i,i} &= \text{diag}(A)_i && (10)\\
A_i{}^j \delta_i{}^j &= (\text{diag}^{-1} \text{diag}\, A)_i{}^j && (11)
\end{aligned}
$$

The Kronecker delta $\delta_i{}^j$ is 1 iff $i = j$ and zero for $i \neq j$.

From the matrix perspective, the first indices index the rows and the second the columns. A second order tensor must have one upstairs

and one downstairs index. The only way of moving downstairs indices upstairs, is to flip ***ALL*** indices, and this does not affect anything.

Summations occur between one down-stairs index and one up-stairs index, (the Einstein convention). Repeated down-stairs or up-stairs indices imply a Hadamard or entry-wise product. As an example, the Hadamard product between two vectors (of the same size) is: $\mathbf{a} \circ \mathbf{b} = [a_1, a_2, ..., a_I]^T . * [b_1, b_2, ..., b_I]^T = [a_1 b_1, a_2 b_2, ..., a_I b_I]^T$

If we have a bunch of second and/or first order tensors (eg. $S_i^{\ j} = B_l^{\ j} W_i^k A^l_{\ k}$), we can convert them into matrix/vector notation using the order of the indices from left to right, and the rule for the transpose.

1. Make the first and the last indices match the LHS ($S_i^{\ j} = W_i^k A^l_{\ k} B_l^{\ j} = (W^T)_i^{\ k} A^l_{\ k} B_l^{\ j}$)
2. Transpose the central objects so the indices run consecutively ($S_i^{\ j} = (W^T)_i^{\ k} (A^T)_k^{\ l} B_l^{\ j}$)
3. Replace with matrix notation ($S = W^T A^T B$)

## 2 Basic derivatives

To convert derivatives found using the suffix notation into matrix derivatives, we need to be aware of one more convention.

Imagine differentiating a vector $\mathbf{x}$ by another vector $\mathbf{y}$. The result is a matrix, but we must choose which way round we want the rows and columns. Conventionally the choice is:

$$\frac{\partial x_i}{\partial y_j} = \Delta_i^{\ j} \tag{12}$$

So the chain rule is easy to apply (and intuitive) by a right hand multiplication:

$$\frac{\partial x_i}{\partial z_k} = \frac{\partial x_i}{\partial y_j}\frac{\partial y_j}{\partial z_k} \tag{13}$$

$$= \Delta_i^{\ j} \Delta_j^{\ k} \tag{14}$$

We might also want to differentiate a matrix by a matrix. Although the resulting forth order tensor cannot be represented a matrix, the object could be required in applying the chain rule (see the example in the following section where we differentiate an object like $\mathrm{tr}(WW^T)$ with

respect to $W$) and so we need rules for assigning the indices. Luckily we have enough conventions to unambiguously specify this:

$$\frac{\partial A_i^{\ j}}{\partial B_k^{\ l}} = \Delta_{i,l}^{j,k} \tag{15}$$

Where the ordering withing the upstairs and downstairs slots is arbitrary.

Note that this relation defines all the derivatives you can form with $2^{nd}$ order tensors and lower (a subset of which are the three types of derivatives that can be represented as matrices).

### 2.0.1 Some examples

Here are some explicit examples:

Find:

$$\frac{\partial \operatorname{tr}(AWCW^TB)}{dW} = \frac{\partial f}{dW} \tag{16}$$

Solution:

$$\begin{aligned}
\frac{\partial f}{dW_i^{\ j}} &= A_k^{\ l} W_l^{\ m} C_m^{\ n} (W^T)_n^{\ p} B_p^k && (17)\\
&= A_k^{\ l} W_l^{\ m} C_m^{\ n} W^p_{\ n} B_p^{\ k} && (18)\\
&= A_k^{\ l} \delta_l^{\ i} \delta_j^{\ m} C_m^{\ n} W^p_{\ n} B_p^{\ k} && (19)\\
&\quad + A_k^{\ l} W_l^{\ m} C_m^{\ n} \delta^{ip} \delta_{jn} B_p^{\ k} && (20)\\
&= A_k^{\ i} C_j^{\ n} W^p_{\ n} B_p^{\ k} && (21)\\
&\quad + A_k^{\ l} W_l^{\ m} C_m^{\ j} B_i^{\ k} && (22)\\
&= C_j^{\ n} (W^T)_n^{\ p} B_p^k A_k^{\ i} && (23)\\
&\quad + (C^T)^j_{\ m} (W^T)^m_{\ l} (A^T)^l_{\ k} (B^T)^k_{\ i} && (24)\\
&= (CW^TBA + C^TW^TA^TB^T)^j_{\ i} && (25)
\end{aligned}$$

Note the discrepancy with some texts (for example the widely used `the matrix cook book`): their results differ by a transpose as their definitions for $\partial x/\partial M$ and $\partial M/\partial x$ are not consistent with regard to their use in the chain rule.

Find:

$$\frac{dA^{-1}}{dx} \tag{26}$$

Solution:

$$\frac{dI_i^{\ j}}{dx} = \frac{dA_i^{\ k}(A^{-1})_k^{\ j}}{dx} \tag{27}$$

$$0 = \frac{dA_i^{\ k}}{dx}(A^{-1})_k^{\ j} + A_i^{\ k}\frac{d(A^{-1})_k^{\ j}}{dx} \tag{28}$$

$$\delta_l^{\ k}\frac{d(A^{-1})_k^{\ j}}{dx} = -(A^{-1})_l^{\ i}\frac{dA_i^{\ k}}{dx}(A^{-1})_k^{\ j} \tag{29}$$

$$\frac{d(A^{-1})_l^{\ j}}{dx} = -\left[A^{-1}\frac{dA}{dx}A^{-1}\right]_l^{\ j} \tag{30}$$

## 3 Differentiating determinants

*complete this section*

## 4 Differentiating structured matrices

Imagine we want to differentiate a matrix with some structure, for example a symmetric matrix. To form the derivatives we can use the matrix-self-derivative:

$$\Gamma_{i,l}^{j,k} = \frac{\partial A_i^{\ j}}{\partial A_k^{\ l}} \tag{31}$$

So that:

$$\frac{\partial f}{\partial A_k^{\ l}} = \frac{\partial f}{\partial A_i^{\ j}}\frac{\partial A_i^{\ j}}{\partial A_k^{\ l}} \tag{32}$$

When forming differentials we have to be careful to **only sum over unique entries** in $A$:

$$df = \frac{\partial f}{\partial A_k^{\ l}}dA_k^{\ l} \tag{33}$$

$$= \sum_{unique\ k,l}\frac{\partial f}{\partial A_k^{\ l}}dA_k^{\ l} \tag{34}$$

One ubiquitous form of structured matrices are the symmetric matrices. For this class the matrix-self-derivative is:

$$\frac{\partial S_i^{\ j}}{\partial S_k^{\ l}} = \delta_i^{\ k}\delta_l^{\ j} + \delta_{i,l}\delta^{j,k} - \delta_i^{\ j}\delta_l^{\ k}\delta_i^{\ k} \tag{35}$$

The first two terms make sure we count the off diagonal elements twice, and the second term avoids over counting of the diagonal and involves some entry-wise products.

### 4.1 A sermon about symmetric matrices

Let's do a family of derivatives correctly that most people muck up.

Find: $\frac{df(S)}{dS}$ where $S$ is symmetric.

$$\begin{aligned}
\frac{\partial f(S)}{\partial S_k^{\ l}} &= \frac{df(S)}{dS_i^{\ j}}\frac{dS_i^{\ j}}{dS_k^{\ l}} && (36)\\
&= f'(S)^i_{\ j}\left[\delta_i^{\ k}\delta_l^{\ j} + \delta_{i,l}\delta^{j,k} - \delta_i^{\ j}\delta_l^{\ k}\delta_i^{\ k}\right] && (37)\\
&= f'(S)^i_{\ j}\left[\delta_i^{\ k}\delta_l^{\ j} + \delta_{i,l}\delta^{j,k} - \delta_i^{\ j}\delta_l^{\ k}\delta_i^{\ k}\right] && (38)\\
&= f'(S)^k_{\ l} + f'(S)_l^{\ k} - f'(S)^i_{\ j}\delta_i^{\ j}\delta_l^{\ k}\delta_i^{\ k} && (39)\\
&= f'(S)^k_{\ l} + f'(S)_l^{\ k} - f'(S)^i_{\ i}\delta_l^{\ k}\delta_i^{\ k} && (40)\\
&= f'(S)^k_{\ l} + f'(S)_l^{\ k} - f'(S)^k_{\ k}\delta_l^{\ k} && (41)\\
&= \left[f'(S) + f'(S)^T - f'(S)\circ I\right]^k_{\ l} && (42)\\
&= \left[2f'(S) - f'(S)\circ I\right]^k_{\ l} && (43)
\end{aligned}$$

If we want the differential, we must sum over all the unique elements (and no more):

$$df(S) = \sum_{i,j\geq i}\left[2f'(S) - f'(S)\circ I\right]^i_{\ j} dS_i^{\ j} \tag{44}$$

Here's a concrete example where people forget the above: imagine finding the ML covariance matrix of a multivariate Gaussian distribution, given some data. The likelihood is given by:

$$\begin{aligned}
P(\{\mathbf{x}_n\}_{n=1}^N) &= \prod_n \det(2\pi\Sigma)^{-1/2}\exp\left[-\frac{1}{2}(\mathbf{x}_n-\mu)^T\Sigma^{-1}(\mathbf{x}_n-\mu)\right] && (45)\\
\log P(\{\mathbf{x}_n\}_{n=1}^N) &= -\frac{N}{2}\left[D\log(2\pi) - \log\det(\Sigma^{-1}) + \mathrm{tr}\left(\Sigma^{-1}X\right)\right] && (46)
\end{aligned}$$

Where we have introduced $X = \left\langle(\mathbf{x}_n-\mu)(\mathbf{x}_n-\mu)^T\right\rangle$. Now we differentiate this wrt. $\Sigma^{-1}$:

$$\frac{\log P(\{\mathbf{x}_n\}_{n=1}^N)}{d\Sigma^{-1}} = -\frac{N}{2}\left[\frac{\log\det(\Sigma^{-1})}{d\Sigma^{-1}} + \frac{\mathrm{tr}\,(\Sigma^{-1}X)}{d\Sigma^{-1}}\right] \quad (47)$$

$$= -\frac{N}{2}\left[2\Sigma - \Sigma\circ I + 2X - X\circ I\right] \quad (48)$$

These are the correct derivatives. As $2\Sigma - \Sigma\circ I + 2X - X\circ I \propto \Sigma - X$ people can recover $\Sigma = X$ without using the symmetrised expressions. However, if we wanted to do gradient ascent here, then we should use the following updates:

$$(\Sigma_{n+1}^{-1})_i^{\ j} = (\Sigma_{n+1}^{-1})_i^{\ j} - \eta\left[2\Sigma - \Sigma\circ I + 2X - X\circ I\right]_i^{j\geq i} \quad (49)$$

Where $\Sigma$ is an upper triangular matrix. Most people would use:

$$(\Sigma_{n+1}^{-1})_i^{\ j} = (\Sigma_{n+1}^{-1})_i^{\ j} - \eta[\Sigma - X]_i^{\ j} \quad (50)$$

If we initialise with symmetric $\Sigma_0^{-1}$ then the incorrect proceedure will walk us along the manifold of symmetric matrices towards the ML solution. You might think numerical errors could, in principle, step us off the manifold of symmetric matrices: Afterall, $(\Sigma^{-1})_i^{\ j}X_j^{\ i}$ is invariant so long as $(\Sigma^{-1})_i^{\ j} + (\Sigma^{-1})_j^{\ i} = \text{const}$. There appears to be no pressure to keep $\Sigma^{-1}$ symmetric. Things actually turn out to be *worse* than this. 45 is not actually a correct expression for a Gaussian distribution if $\Sigma^{-1}$ is not symmetric. Specifically the normalising constant should be replaced by $\det(\frac{1}{2}[\Sigma+\Sigma^T])$. If you don't use this symmetrised form, the manifold of symmetric matrices lies along a **minimum** and the **ML solution is a saddle point** (see Fig. 1). For this reason, it seems prudent to use the correct gradients when doing gradient ascent, and to remember to normalise correctly.

## 5 Relation to the vec and kronecker product operators

The vec, Kronecker product, vector transpose, and reshape operators shuffle tensors so they can be represented in arrays of different dimensionalities (and sizes). The are most intuitively defined by visual examples, but useful results can be proved using a tensor representation that always involves a tensor product between the object we are transforming and an indicator tensor.

In this section our tensor algebra does not need to deal with entry-wise products etc.. Therefore, to aid the clarity of the presentation we

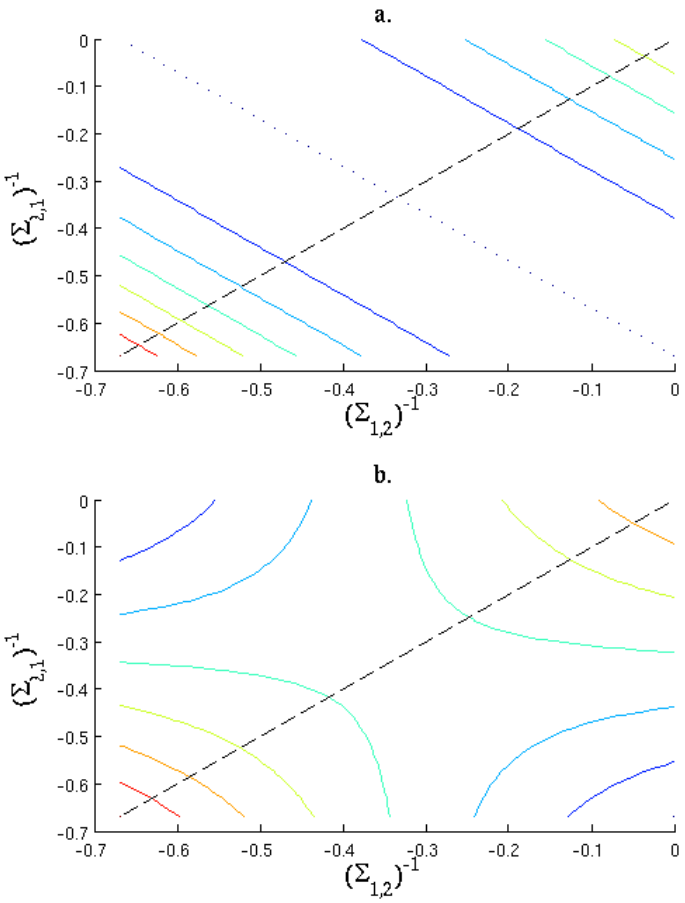


Figure 1: a. The expected cost function is symmetric, and the maximum is a ridge. b. The cost function with the incorrect normaliser is not symmetric and the MLparameter values correspond to a saddle point.

use the more usual suffix notation. The results presented here are easy to generalise using the previous framework (as is needed to relate the entry-wise and Kronecker products, and diag, for example), but the result is less aesthetic.

### 5.1 Vec

The vec operator lets you represent a matrix as a vector, by stacking the columns. For example:

$$\operatorname{vec}\left[\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{pmatrix}\right] = \begin{pmatrix} x_{11} \\ x_{21} \\ x_{12} \\ x_{22} \\ x_{12} \\ x_{23} \end{pmatrix} \tag{51}$$

The tensor representation of this operator is:

$$x_i = V_{iab} X_{ab} \tag{52}$$
$$V_{iab} = \delta_{i,a+(b-1)A} \tag{53}$$

### 5.2 Kronecker Product

The Kronecker product operator ($\otimes$) lets you represent the outer product of two matrices (a 4th order tensor) as a matrix.

$$\begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \otimes Y = \begin{pmatrix} x_{11}Y & x_{12}Y \\ x_{21}Y & x_{22}Y \end{pmatrix} \tag{54}$$

Alternatively, written as a tensor, we have:

$$Z_i^j = K_{iac}^{jbd} X_b^a Y_d^c \tag{55}$$
$$K_{ijabcd} = \delta_{i,c+(a-1)C} \delta_{j,d+(b-1)D} \tag{56}$$

Examples The important result relating the vec and Kronecker products is:

$$(C^T \otimes A)\operatorname{vec}(B) = \operatorname{vec}(ABC) \tag{57}$$

which can be proved using the definitions above:

$$
\begin{aligned}
[(C^T \otimes A)\,\mathrm{vec}(B)]_{ij} &= K_{ijabcd} C_{ba} A_{cd} V_{jef} B_{ef} && (58)\\
&= \delta_{i,c+(a-1)C} \delta_{j,d+(b-1)D} C_{ba} A_{cd} \delta_{j,e+(f-1)E} B_{ef} && (59)\\
&= \delta_{i,c+(a-1)C} \delta_{d,e} \delta_{bf} C_{ba} A_{cd} B_{ef} && (60)\\
&= \delta_{i,c+(a-1)C} C_{ba} A_{cd} B_{bd} && (61)\\
&= V_{ica} A_{cd} B_{bd} C_{ba} && (62)\\
&= \mathrm{vec}(ACB) && (63)
\end{aligned}
$$

### 5.3 vec-transpose

This is not the same as reshape in `MATLAB` despite what Tom Minka claims. *complete this section*

### 5.4 reshape

Reshape generalises the vec operator. It allows us to handle high dimensional objects easily. For example, by remapping tensors into matrices we can form tensor inverses.

We want to be able to map an array of size $A \times B \times C \times \ldots$ with in total $A \times B \times C \times \ldots = N$ elements into another array of $I \times J \times K \times \ldots = N$ elements. To do this we need to specify where each of the elements in the old array appears in the new array. There are lots of choices for how we lay down the elements in the new array. It would be useful to do this systematically.

One way to do this is to come up with a systematic method for numbering each element in an array, and then we could lay down elements in the new array such that they will be assigned the same number. A systematic numbering system can be constructed as follows.

In a normal counting system, we choose a base $B$ (eg. 10, 2). We can uniquely form integer numbers as a sum of upto $B-1$ 1s, $B$s, $B^2$s and so on. eg. $B = 10$, $954 = 9 \times 100 + 5 \times 50 + 4 \times 1$ eg. $B = 2$, $13 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 1$. The coefficients are the representation of the number. Let's define a new counting system which has a non-constant base. The last number $i$ will again represent the number of ones and will take a values from 0 to $I-1$, the second number $j$ represents the number of $I$s and takes values from 0 to $J-1$, the third number $k$ represents the number of $I \times J$s and takes values from 0 to $K-1$, and so on. eg. Using $\{I, J, K\} = \{2, 3, 4\}$ the number 21 is $3 \times 6 + 1 \times 2 + 1 \times 1$ and we can represent $I \times J \times K = 24$ numbers this way. Now if we associate $i+1, j+1, k+1, \ldots$ with the position in

an $N$ dimensional array, we have assigned all points in that array with a unique integer number that forms a sequence.

$$\begin{aligned} T_{i,j,k,\ldots} &= R^{I,J,K,\ldots;A,B,C,\ldots}_{i,j,k,\ldots,a,b,c,\ldots} S_{a,b,c,\ldots} && (64)\\ &= \text{reshape}(S,[I,J,K,\ldots]) && (65)\\ R^{I,J,K,\ldots;A,B,C,\ldots}_{i,j,k,\ldots,a,b,c,\ldots} &= \delta_{a-1+(b-1)A+(c-1)AB+\ldots,i-1+(j-1)I+(k-1)IJ+\ldots} && (66) \end{aligned}$$

### 5.4.1 Examples:

Here are some examples and useful results:

From this it is simple to show that reshaping a tensor, and then reshaping it back to its original size returns all the entries to their original positions:

$$\begin{aligned} T_{a',b',c',\ldots} &= R^{A,B,C,\ldots;I,J,K,\ldots}_{a',b',c',\ldots,i,j,k,\ldots} R^{I,J,K,\ldots;A,B,C,\ldots}_{i,j,k,\ldots,a,b,c,\ldots} S_{a,b,c,\ldots} && (67)\\ &= \delta_{a'-1+(b'-1)A+(c'-1)AB+\ldots,i-1+(j-1)I+(k-1)IJ+\ldots} \\ &\qquad \delta_{a-1+(b-1)A+(c-1)AB+\ldots,i-1+(j-1)I+(k-1)IJ+\ldots} S_{a,b,c,\ldots} && (68)\\ &= \delta_{a'-1+(b'-1)A+(c'-1)AB+\ldots,a-1+(b-1)A+(c-1)AB+\ldots} S_{a,b,c,\ldots} && (69)\\ &= \delta_{a',a}\delta_{b',b}\delta_{c',c} S_{a,b,c,\ldots} && (70)\\ &= S_{a',b',c',\ldots} && (71) \end{aligned}$$

This result should be obvious. The way we introduced the reshape operator was via a numbering system that was unique for all arrays of a given shape. This means if we reshape an array and reshape it again, the result must be equivalent to reshaping directly: intermediate reshapings cannot effect anything.

A problem where the reshape operator is useful arises in multi-linear models. There we have to solve:

$$\alpha_{d,i,j,\ldots} = g_{d,a,b,\ldots}\beta_{a,b,\ldots,i,j,\ldots} \qquad (72)$$

where we want $g_{d,a,b,\ldots}$ and we know $\alpha_{d,i,j,\ldots}$ and $\beta_{a,b,\ldots,i,j,\ldots}$. The dimensionalities are $I = A, J = B \ldots$. The solution amounts to finding the inverse of $\beta_{a,b,\ldots,i,j,\ldots}$. Reshaping the left and right hand sides into $[D, Q = I \times J \times \ldots]$ matrices we have:

$$\begin{aligned} R^{D,Q;D,I,J,\ldots}_{e,q,d,i,j,\ldots}\alpha_{d,i,j,\ldots} &= R^{D,Q;I,J,\ldots}_{e,q,d,i,j,\ldots} g_{d,a,b,\ldots}\beta_{a,b,\ldots,i,j,\ldots} && (73)\\ &= \delta_{e+(q-1)D,d+(i-1)D+(j-1)DI+\ldots} g_{d,a,b,\ldots}\beta_{a,b,\ldots,i,j,\ldots} && (74)\\ &= \delta_{e,d}\delta_{q,i+(j-1)I+\ldots} g_{d,a,b,\ldots}\beta_{a,b,\ldots,i,j,\ldots} && (75)\\ &= g_{e,a,b,\ldots}\delta_{q,i+(j-1)I+\ldots}\beta_{a,b,\ldots,i,j,\ldots} && (76) \end{aligned}$$

Letting $X = \text{reshape}(\alpha, [D, Q])$, we replace the tensor product on the RHS with a matrix product using the reshape operator again:

$$\begin{aligned}
X_{e,q} &= \delta_{a',a}\delta_{b',b}\ldots g_{e,a,b,\ldots}\delta_{q,i+(j-1)I+\ldots}\beta_{a',b',\ldots,i,j,\ldots} && (77)\\
&= R^{A,B,\ldots;Q}_{a',b',\ldots,g} R^{Q;A,B,\ldots}_{g,a,b,\ldots} \\
&\quad g_{e,a,b,\ldots}\delta_{q,i+(j-1)I+\ldots}\beta_{a',b',\ldots,i,j,\ldots} && (78)\\
&= R^{Q;A,B,\ldots}_{g,a,b,\ldots} g_{e,a,b,\ldots} \\
&\quad \delta_{q,i+(j-1)I+\ldots} R^{A,B,\ldots;Q}_{a',b',\ldots,f,g}\beta_{a',b',\ldots,i,j,\ldots} && (79)\\
&= R^{D,Q;D,A,B,\ldots}_{e,g,e',a,b,\ldots} g_{e',a,b,\ldots} \\
&\quad R^{Q,Q;A,B,\ldots,A,B,\ldots}_{g,q,a',b',\ldots,i,j\ldots}\beta_{a',b',\ldots,i,j,\ldots} && (80)\\
&= \text{reshape}(g, [D, Q])_{e,g}\,\text{reshape}(\beta, [Q, Q])_{g,q} && (81)
\end{aligned}$$

Letting $Y = \text{reshape}(\beta, [Q, Q])$ the solution is:

$$g_{d,a,b,\ldots} = \text{reshape}(XY^{-1}, [D, A, B, \ldots])_{d,a,b,\ldots} \quad (82)$$