tf.contrib.distributions.matrix_diag_transform

```
matrix_diag_transform(
    matrix,
    transform=None,
    name=None
)
```

Defined in `tensorflow/python/ops/distributions/util.py`.

See the guide: Statistical Distributions (contrib) > Multivariate distributions

Transform diagonal of [batch-]matrix, leave rest of matrix unchanged.

Create a trainable covariance defined by a Cholesky factor:

```
# Transform network layer into 2 x 2 array.
matrix_values = tf.contrib.layers.fully_connected(activations, 4)
matrix = tf.reshape(matrix_values, (batch_size, 2, 2))

# Make the diagonal positive. If the upper triangle was zero, this would be a
# valid Cholesky factor.
chol = matrix_diag_transform(matrix, transform=tf.nn.softplus)

# LinearOperatorTriL ignores the upper triangle.
operator = LinearOperatorTriL(chol)
```

Example of heteroskedastic 2-D linear regression.

```
# Get a trainable Cholesky factor.
matrix_values = tf.contrib.layers.fully_connected(activations, 4)
matrix = tf.reshape(matrix_values, (batch_size, 2, 2))
chol = matrix_diag_transform(matrix, transform=tf.nn.softplus)

# Get a trainable mean.
mu = tf.contrib.layers.fully_connected(activations, 2)

# This is a fully trainable multivariate normal!
dist = tf.contrib.distributions.MVNCholesky(mu, chol)

# Standard log loss. Minimizing this will "train" mu and chol, and then dist
# will be a distribution predicting labels as multivariate Gaussians.
loss = -1 * tf.reduce_mean(dist.log_prob(labels))
```

Args:

- `matrix`: Rank `R` `Tensor`, `R >= 2`, where the last two dimensions are equal.
- `transform`: Element-wise function mapping `Tensors` to `Tensors`. To be applied to the diagonal of `matrix`. If `None`, `matrix` is returned unchanged. Defaults to `None`.
- `name`: A name to give created ops. Defaults to "matrix_diag_transform".

Returns:

A `Tensor` with same shape and `dtype` as `matrix`.
