Supplementary appendices for “Connecting psychophysical performance to neuronal response properties II: Contrast decoding and detection”

Keith A. May
Joshua A. Solomon

Appendix A: Main symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>Base of logarithm when Michelson contrast, $c$, is expressed as log contrast, $x$</td>
</tr>
<tr>
<td>$c$</td>
<td>Michelson contrast</td>
</tr>
<tr>
<td>$c_{1/2}$</td>
<td>Naka-Rushton function parameter: semi-saturation contrast</td>
</tr>
<tr>
<td>$c_{1/3}$</td>
<td>The Michelson contrast for which the output of the Naka-Rushton function exceeds $r_0$ by $r_{\text{max}}/3$</td>
</tr>
<tr>
<td>$H$</td>
<td>A function (used with subscripts) that appears in the approximations of the Fisher information (see Appendix E)</td>
</tr>
<tr>
<td>$J$</td>
<td>Fisher information</td>
</tr>
<tr>
<td>$j$</td>
<td>Integer index of the neurons in a population</td>
</tr>
<tr>
<td>$K$</td>
<td>The number of neurons being monitored by the observer</td>
</tr>
<tr>
<td>$N$</td>
<td>Random variable representing the number of spikes produced by a neuron</td>
</tr>
<tr>
<td>$n$</td>
<td>The value of $N$ on a particular stimulus presentation</td>
</tr>
<tr>
<td>$\mathbf{N}$</td>
<td>Vector of random variables representing the number of spikes produced by each neuron in the population</td>
</tr>
<tr>
<td>$\mathbf{n}$</td>
<td>The value of $\mathbf{N}$ on a particular stimulus presentation</td>
</tr>
<tr>
<td>$P$</td>
<td>Probability</td>
</tr>
<tr>
<td>$q$</td>
<td>Naka-Rushton function parameter: exponent, which determines tuning sharpness</td>
</tr>
<tr>
<td>$R$</td>
<td>Random variable representing the mean response of a neuron</td>
</tr>
<tr>
<td>$r(x)$</td>
<td>The neuron’s tuning function, which gives the value of $R$ on a particular stimulus presentation</td>
</tr>
<tr>
<td>$r_0$</td>
<td>Naka-Rushton function parameter: spontaneous firing rate</td>
</tr>
<tr>
<td>$r_{\text{max}}$</td>
<td>Naka-Rushton function parameter: maximum increment from $r_0$</td>
</tr>
<tr>
<td>$T$</td>
<td>Number of trials</td>
</tr>
<tr>
<td>$u_n$</td>
<td>Parameter of the “approximate” probability distribution used in Appendix E</td>
</tr>
<tr>
<td>$v$</td>
<td>Parameter that appears in the expressions for the “approximate” probability distribution (Appendix E) and $\hat{f}(x)$</td>
</tr>
<tr>
<td>$X$</td>
<td>Random variable representing the stimulus log Michelson contrast</td>
</tr>
<tr>
<td>$x$</td>
<td>The value of $X$ on a particular stimulus presentation</td>
</tr>
<tr>
<td>$\hat{X}$</td>
<td>Random variable representing the estimated stimulus level after decoding the spike counts</td>
</tr>
<tr>
<td>$\hat{x}$</td>
<td>The value of $\hat{X}$ for a particular stimulus presentation</td>
</tr>
<tr>
<td>$z$</td>
<td>Naka-Rushton function parameter: log semi-saturation contrast</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Weibull “threshold” parameter</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Weibull “slope” or “shape” parameter</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Weibull “lapse rate” parameter</td>
</tr>
</tbody>
</table>
Appendix B: Finite series expansion of the Tolhurst likelihood function

The finite series expansion of the Tolhurst likelihood function is derived in Theorem 1, below. As a prelude to this theorem, we will find it useful to define a function, \( \Omega_n(\bar{\xi}) \), given by

\[ \Omega_n(\bar{\xi}) = \frac{1}{e^{\bar{\xi}}} \sum_{\mu=0}^{\infty} \frac{\mu^n}{\mu!} \bar{\xi}^\mu. \]  

First, consider the case of \( n = 0 \). In this case, we have

\[ \Omega_0(\bar{\xi}) = \frac{1}{e^{\bar{\xi}}} \sum_{\mu=0}^{\infty} \frac{1}{\mu!} \bar{\xi}^\mu. \]  

The summation in Equation (B.2) is the series expansion of \( e^{\bar{\xi}} \), and so we have

\[ \Omega_0(\bar{\xi}) = 1. \]  

Lemma 1 derives a recursive expression for \( \Omega_n(\bar{\xi}) \) for any integer \( n > 0 \).

**Lemma 1.** For all integers \( n > 0 \),

\[ \Omega_n(\bar{\xi}) = \frac{\bar{\xi}}{e^{\bar{\xi}}} \sum_{v=0}^{\infty} \sum_{k=0}^{n-1} \frac{(n-1)}{v!} \left( \frac{1}{\mu!} \bar{\xi}^\mu \right)^v. \]  

**Proof.** For \( n > 0 \), the first term of the infinite series expansion of \( \Omega_n(\bar{\xi}) \) given by Equation (B.1) is 0, because this term includes multiplication by \( \mu^n \), which is 0 for the first term (since \( \mu = 0 \)). Thus, we can start the series at \( \mu = 1 \):

\[ \Omega_n(\bar{\xi}) = \frac{1}{e^{\bar{\xi}}} \sum_{\mu=1}^{\infty} \frac{\mu^n}{\mu!} \bar{\xi}^\mu. \]  

Now, since \( \mu \neq 0 \) in Equation (B.4), we can divide the top and bottom of each term in the series by \( \mu \) to give

\[ \Omega_n(\bar{\xi}) = \frac{1}{e^{\bar{\xi}}} \sum_{\mu=1}^{\infty} \frac{\mu^{n-1}}{(\mu-1)!} \bar{\xi}^\mu. \]  

Let us define \( \nu \) as

\[ \nu = \mu - 1. \]  

Then,

\[ \Omega_n(\bar{\xi}) = \frac{1}{e^{\bar{\xi}}} \sum_{v=0}^{\infty} \sum_{\nu=0}^{\infty} \frac{(v+1)^{n-1}}{v!} \bar{\xi}^{v+1}. \]  

Since \( n > 0 \), we can expand \( (v+1)^{n-1} \) in Equation (B.8) using the binomial expansion:

\[ \Omega_n(\bar{\xi}) = \frac{\bar{\xi}}{e^{\bar{\xi}}} \sum_{v=0}^{\infty} \sum_{\nu=0}^{\infty} \frac{n-1}{v!} \left( \frac{1}{\mu!} \bar{\xi}^\mu \right)^v. \]  

Applying the distributive rule [i.e. \( a(b + c) = ab + ac \)] to Equation (B.9), we get

\[ \Omega_n(\bar{\xi}) = \frac{\bar{\xi}}{e^{\bar{\xi}}} \sum_{v=0}^{\infty} \sum_{k=0}^{n-1} \frac{(n-1)}{v!} \left( \frac{1}{\mu!} \bar{\xi}^\mu \right)^v. \]
Rearranging the order of summation gives

\[
\Omega_n(\xi) = \frac{\xi^n}{e^\xi} \sum_{k=0}^{n-1} \sum_{\nu=0}^{\infty} \frac{(n-1)_k}{\nu!} \xi^\nu.
\]  
(B.11)

Applying the distributive rule again gives

\[
\Omega_n(\xi) = \xi \sum_{k=0}^{n-1} \left[ \frac{(n-1)_k}{k!} \sum_{\nu=0}^{\infty} \frac{\xi^\nu}{\nu!} \right] \Omega_k(\xi). \quad \square 
\]  
(B.12)

The finite series expansion of each \(\Omega_n(\xi)\) is a polynomial of degree \(n\). As examples, Equations (B.14) to (B.19) give full, non-recursive expressions for \(\Omega_n(\xi)\) for \(n = 0\) to \(5\).

\[
\Omega_0(\xi) = 1 \quad \text{(B.14)}
\]

\[
\Omega_1(\xi) = \xi \quad \text{(B.15)}
\]

\[
\Omega_2(\xi) = \xi^2 + \xi \quad \text{(B.16)}
\]

\[
\Omega_3(\xi) = \xi^3 + 3\xi^2 + \xi \quad \text{(B.17)}
\]

\[
\Omega_4(\xi) = \xi^4 + 6\xi^3 + 7\xi^2 + \xi \quad \text{(B.18)}
\]

\[
\Omega_5(\xi) = \xi^5 + 10\xi^4 + 25\xi^3 + 15\xi^2 + \xi. \quad \text{(B.19)}
\]

The polynomial coefficients for any \(\Omega_n(\xi)\) can be calculated using the MATLAB program, \texttt{OmegaCoefficients}, given in Appendix C.

Lemma 1 shows how to express the infinite series expansion in Equation (B.1) as a finite series, given in Equation (B.13). We now prove Theorem 1, which uses Lemma 1 to express the Tolhurst likelihood function as a finite series. To reduce notational clutter, we will use the single letter \(r\), instead of \(r(x)\), to represent the mean spike rate.

**Theorem 1.**

\[
P_{\text{Tolhurst}}(N = n \mid R = r) = \frac{e^{r(1/e-1)}}{n!} \Omega_n(r/e). \quad \text{(B.23)}
\]

**Proof.** Using Equation (4) of the main paper to expand the expression for the Poisson distribution in Equation (5), we have

\[
P_{\text{Tolhurst}}(N = n \mid R = r) = \sum_{\mu=0}^{\infty} \frac{\mu^n}{n!} e^{-\mu} \frac{r^\mu}{\mu!} e^{-r} \quad \text{(B.20)}
\]

\[
= e^{-r} \sum_{\mu=0}^{\infty} \frac{\mu^n}{\mu!} \left(\frac{r}{e}\right)^\mu \quad \text{(B.21)}
\]

\[
= e^{-r} \frac{e^{r(e-1)}}{n!} \sum_{\mu=0}^{\infty} \frac{\mu^n}{\mu!} \left(\frac{r}{e}\right)^\mu \quad \text{(B.22)}
\]

\[
= e^{r(1/e-1)} \frac{1}{n!} \Omega_n(r/e). \quad \square \quad \text{(B.23)}
\]

Equation (B.23) can be evaluated for any \(r\) and \(n\) using the MATLAB program, \texttt{PTolhurst}, given in Appendix D.

**Appendix C: Software for calculating the coefficients of \(\Omega_n\)**

In this appendix, we give two MATLAB programs for calculating the coefficients of the \(\Omega\) function, \(\Omega_n(\xi)\). The programs both implement the recursive definition of \(\Omega_n(\xi)\), given in Equation (B.13).

The first program, \texttt{OmegaCoefficientsRecursive}, uses a recursive algorithm that outputs the coefficients of \(\Omega_n(\xi)\). The program is easy to follow because it is exactly analogous to the recursive definition of the \(\Omega\) function. However, for large \(n\), it is epically inefficient, as it keeps recalculating the coefficients of \(\Omega\) functions of lower degree.

The second program, \texttt{OmegaCoefficients}, uses a non-recursive algorithm that calculates each set of coefficients once, and then stores them for future use. This program is less easy to follow, as it contains an outer loop, and is not exactly analogous to the recursive definition of the \(\Omega\) function, but it is much more efficient. Because the program stores the coefficients of each \(\Omega\) function from \(\Omega_0(\xi)\) to \(\Omega_n(\xi)\), it outputs the coefficients of all of these Omega functions, rather than just the coefficients of \(\Omega_n(\xi)\).
function C = OmegaCoefficientsRecursive(n)
% Returns C, the vector of coefficients of the Omega function of degree n.
% C has n+1 terms, as the Omega function is a polynomial of degree n.
% n must be a scalar integer.

if n == 0
  % Omega(0, x) = 1 for all x, i.e. a polynomial of degree 0 with coefficient 1
  C = 1;
else
  % set up one coefficient for every term in the n-degree polynomial
  C = zeros(1,n+1);
  for k = 0:(n-1)
    % For each k in the summation in Equation (33), find the
    % coefficients of the Omega function of degree k.
    Ck = OmegaCoefficientsRecursive(k);
    % Then shift the coefficients along one position in the polynomial
    % (i.e. one power of x, because the omega function is multiplied by
    % x in Equation (33), so the coefficient of the mth term from the
    % right in in the old Omega function becomes the coefficient of the
    % (m+1)th term from the right in the new Omega function.
    % Also, pad out slots to the left to make n+1 coefficients.
    % Then multiply by the binomial coefficient, as in Equation (33),
    % and add to the total value of each coefficient.
    C = C + (factorial(n-1) / (factorial(k) * factorial(n-1-k))) * ...
       [zeros([1,n-1-k]), Ck, 0];
  end
end

function C = OmegaCoefficients(n)
% Returns C, the matrix of coefficients of Omega functions of degree 0 to n.
% Row m+1 of C gives the coefficients for the Omega function of degree m
% C has n+1 columns, as the highest-degree Omega function is a polynomial of
% degree n.
% n must be a scalar integer.

C = zeros(n+1,n+1); % n+1 Omega functions (rows) and n+1 coefficients (columns)

% Omega(0, x) = 1 for all x, i.e. a polynomial of degree 0, with coefficient 1
C(1,n+1) = 1;

% calculate coefficients of each omega function of degree 1 to n
for m = 1:n
  for k = 0:(m-1)
    % For each k in the summation in Equation (33), find the
    % coefficients of the Omega function of degree k
    Ck = C(k+1,:);
    % Then shift the coefficients along one position in the polynomial
    % (i.e. one power of x, because the omega function is multiplied by
    % x in Equation (33), so the coefficient of the mth term from the
    % right in in the old Omega function becomes the coefficient of the
    % (m+1)th term from the right in the new Omega function.
    % Then multiply by the binomial coefficient, as in Equation (33),
    % and add to the total value of each coefficient.
    C(m+1,:) = C(m+1,:) + ... 
              (factorial(m-1) / (factorial(k) * factorial(m-1-k))) * ...
              [Ck(2:end), 0];
  end
end
Appendix D: Software for calculating the Tolhurst likelihood function

In this appendix, we give two MATLAB programs for calculating the Tolhurst likelihood function, \( P_{\text{Tolhurst}}(N = n | R = r) \). Each program calls a program that calculates the relevant \( \Omega \) function and then calculates the Tolhurst likelihood function from this as in Equation (B.23) (strictly speaking these “programs” are MATLAB functions, but we use the term “program” here to avoid confusion with the mathematical functions that are being calculated). As with the programs for calculating the coefficients of the \( \Omega \) function, we present recursive and non-recursive versions. Again, the recursive version is easier to follow, but too inefficient to be of practical use other than for small values of \( n \). The recursive version, \( P_{\text{Tolhurst}}^\text{Recursive}(n, r) \), gives \( P_{\text{Tolhurst}}(N = n | R = r) \) for a scalar integer \( n \), and any size matrix of \( r \) values. The non-recursive version, \( P_{\text{Tolhurst}}(N = m | R = r) \), calculates \( P_{\text{Tolhurst}}(N = m | R = r) \) for all integers, \( m \), between 0 and \( n \), inclusive. \( n \) must be a scalar integer. \( r \) must be a real scalar or row vector. Row \( m+1 \) of the output gives \( P_{\text{Tolhurst}}(N = m | R = r) \) for each element of the input argument, \( r \).

```matlab
function P = PTolhurstRecursive(n,r)
    % Returns probability of n for the Tolhurst process, given a mean of r.
    % n must be a scalar integer.
    % r can be a real scalar or matrix of any size or number of dimensions.
    % The output, P, has the same size and number of dimensions as r.
    e = exp(1);
    P = (exp(r*(1/e - 1))/factorial(n)) .* OmegaRecursive(n, r/e);
end

function y = OmegaRecursive(n,x)
    if n == 0
        y = 1;
    else
        y = 0;
        for k = 0:(n-1)
            y = y + (factorial(n-1) / (factorial(k) * factorial(n-1-k))) * OmegaRecursive(k,x);
        end
        y = x .* y;
    end
end

function P = PTolhurst(n,r)
    % Returns probability of 0 to n for the Tolhurst process, given a mean r.
    % n must be a scalar integer.
    % r can be a real scalar or row vector.
    % The (m+1)th row of P gives P(N = m | R = r) for each element in the input argument, r.
    e = exp(1);
    P = exp(repmat(r,[n+1,1])*(1/e - 1)) ./ ...
    repmat(factorial([0:n]'),[1,length(r)]) .* Omega(n,r/e);
end

function y = Omega(n,x)
    % Note, Omega(m,x) is stored in row m+1 or Omegas
    Omegas = zeros(n+1,length(x));
    Omegas{1,:) = ones(1,length(x));
    % calculate Omega(m,:) for each m = 1:n
    for m = 1:n
        k = [0:(m-1)]';
        % Calculate binomial coefficients
        C = (factorial(m-1) ./ (factorial(k) .* factorial(m-1-k)));
        C = repmat(C, [1,length(x)]);
        Omegas(m+1,:) = x .* sum(C .* Omegas([0:(m-1)]+1,:),1);
    end
    y = Omegas;
```
Appendix E: Approximations of the Fisher information for non-Poisson-spiking neurons

A derivation of an exact formula for the Fisher information of a neuron with a Tolhurst or Consul-Jain spiking process turned out to be intractable, so we closely approximated both of these distributions using a different expression, for which an analytical expression for the Fisher information could be derived. We use the term “approximate distribution” to refer to the expression that we used to approximate the Tolhurst and Consul-Jain distributions. The approximate distribution is given by

\[ P_{\text{Approx}}(N = n | R = r(x)) = u_n P_{\text{Poisson}}(N = n/v | R = r(x)/v) \]

\[ = u_n \left( \frac{r(x)/v}{\Gamma(n/v + 1)} \right)^{n/v} \exp(-r(x)/v) \]

In Equation (E.2), we have to express the Poisson distribution using the continuous gamma function, \( \Gamma(n/v + 1) \), instead of the factorial, \( (n/v)! \), so that it can be evaluated for non-integer values of \( n/v \). This approximate distribution has a parameter \( v \), and an infinite set of parameters, \( u_n \), one for each possible value of \( n \) (the integers 0 to \( \infty \)). However, it turns out that the values of the \( u_n \) parameters have no effect on the Fisher information (this comes about because the Fisher information is the average negative 2nd derivative of the log-likelihood function – the log function converts multiplicative constants, \( u_n \), into additive constants, which have no effect on the derivative). Thus, we can proceed as if \( P_{\text{Approx}} \) was parameterized only by \( v \).

The only purpose of fitting the \( u_n \) parameters is to verify that \( P_{\text{Approx}} \) is good approximation. Note that, for many values of \( v \), it may be impossible to find a set of parameters, \( u_n \), for which \( \sum_{n=0}^{\infty} P_{\text{Approx}}(N = n | R = r(x)) \) is exactly equal to 1 for all \( r(x) \), and so \( P_{\text{Approx}} \) is not technically a probability distribution. However, it can provide a sufficiently close fit to the true Tolhurst and Consul-Jain distributions for it to generate a good approximation of the Fisher information.

Appendix F shows that, assuming \( P_{\text{Approx}} \) is a probability distribution generating a neuron’s spikes, the Fisher information, \( J \), is given by

\[ J = \frac{1}{v} \frac{r'(x)^2}{r(x)}. \]

\textbf{Fisher information for the Tolhurst process}

Using Equation (B.14) of Appendix B to substitute for \( \Omega_n(\cdot) \) in Equation (B.23) when \( n = 0 \), we obtain

\[ P_{\text{Tolhurst}}(N = 0 | R = r(x)) = e^{(1/e-1)r(x)} \]

\[ = P_{\text{Poisson}}(N = 0 | R = (1-1/e)r(x)), \]

which has the form of \( P_{\text{Approx}} \) with \( v = 1/(1-1/e) \). For \( n > 0 \), we cannot express the Tolhurst distribution exactly in the form of \( P_{\text{Approx}} \), but we can closely approximate it for all \( n \) except \( n = 1 \). For \( n > 1 \), we have found that

\[ P_{\text{Tolhurst}}(N = n | R = r(x)) \approx u_n P_{\text{Poisson}}(N = n/2 | R = r(x)/2), \]

\begin{table}[h]
\begin{tabular}{|c|c|}
\hline
\( n \) & \( u_n \) \\
\hline
2 & 0.5007 \\
3 & 0.5024 \\
4 & 0.5018 \\
5 & 0.5015 \\
6 & 0.5014 \\
7 & 0.5012 \\
8 & 0.5011 \\
9 & 0.5010 \\
10 & 0.5009 \\
11 & 0.5008 \\
\hline
\end{tabular}
\end{table}

\textbf{Table E.1. Parameters that minimize the error in the approximation of the Tolhurst distribution given by Relation (E.6). These parameters were fitted by using Equation (E.2) to evaluate the right hand side of Relation (E.6) over a range of \( r(x) \) from 0 to 100 in steps of 0.01, and comparing with the true Tolhurst distribution (the left hand side of Relation (E.6), given by Equation (B.23)) for the same values of \( r(x) \). We found the \( u_n \) parameters that minimized the sum of squared differences between the left and right sides of Relation (E.6). All decimal expansions are shown to an accuracy of 4 significant figures.}
Figure E.1. The Tolhurst likelihood functions (thin, coloured curves), plotted against the best-fitting approximations (thick, black curves). \( r(x) \) is the mean number of spikes elicited by stimulus \( x \). For \( n = 1 \), the approximation is described by Relation (E.7). For \( n > 1 \), the approximation is described by Relation (E.6) with parameter values, \( u_n \), given in Table E.1.

with best-fitting \( u_n \) values given in Table E.1 for \( n = 2 \) to 11. The approximation of \( P_{\text{Tolhurst}} \) in Relation (E.6) has the form of \( P_{\text{Approx}} \) with \( v = 2 \). For \( n = 1 \), there is no really good approximation of the Tolhurst process that has the form of \( P_{\text{Approx}} \); the best (least-squares) approximation is

\[ P_{\text{Tolhurst}}(N=1|R=r(x)) \approx 0.4106 \times P_{\text{Poisson}}(N=1/2.306|R=r(x)/2.306), \] (E.7)

which has the form of \( P_{\text{Approx}} \) with \( v = 2.306 \). Figure E.1 plots the true Tolhurst likelihood functions against the approximations given by Relations (E.6) and (E.7).

So, for a proportion \( P(N=0) \) of trials, the Tolhurst distribution is \( P_{\text{Approx}} \) with \( v = 1/(1-1/e) \), for which the Fisher information (given by Equation (E.3)) is

\[ (1-1/e)r'(x)^2/r(x); \]

for a proportion \( P(N=1) \) of trials, the Tolhurst distribution is reasonably well approximated by \( P_{\text{Approx}} \) with \( v = 2.306 \), for which the Fisher information is

\[ r'(x)^2/[2r(x)]; \]

for the remaining proportion of trials, \([1-P(N=0)-P(N=1)]\), the Tolhurst distribution is well approximated by \( P_{\text{Approx}} \) with \( v = 2 \), for which the Fisher information is

\[ r'(x)^2/[2r(x)]. \]

Since the Fisher information is a trial-averaged quantity (the average negative 2nd derivative of the log-likelihood function), we can calculate it separately for these three different scenarios (\( n = 0, n = 1 \) and \( n > 1 \)), and then average them, weighted according to their probabilities, to give a close approximation of the Fisher information for a single Tolhurst-spiking neuron:

\[ J \approx \left( \frac{1}{1-1/e}P(N=0) + \frac{P(N=1)}{2.306} + \frac{1-P(N=0)-P(N=1)}{2} \right) \frac{r'(x)^2}{r(x)}. \] (E.8)
Equation (E.4) gives an expression for \( P(N=0) \) for the Tolhurst distribution. Using Equation (B.15) of Appendix B to substitute for \( \Omega_n(x) \) in Equation (B.23) when \( n=1 \), we get an expression for \( P(N=1) \):

\[
P_{\text{Tolhurst}}(N=1 | R = r(x)) = r(x)e^{(1/e-1)r(x)-1}.
\]  
(E.9)

Using Equations (E.4) and (E.9) to substitute for \( P(N=0) \) and \( P(N=1) \) in Relation (E.8), we obtain

\[
J \approx H_{\text{Tolhurst}}(x) \times \frac{r'(x)^2}{r(x)}, 
\]
(E.10)

where

\[
H_{\text{Tolhurst}}(x) = \left( \frac{1}{2} - \frac{1+0.0663 \times r(x)}{e} \right) \times \exp[r(x)(1/e-1)] + \frac{1}{2}.
\]  
(E.11)

For a population of statistically independent neurons, the Fisher information is the sum of the Fisher information of the individual neurons. This gives us a good approximation of the decoding precision for a population of independent Tolhurst-spiking neurons:

\[
\tau_{\text{Tolhurst}}(x) = \sum_{j=1}^{K} H_{\text{Tolhurst}}(x) \times \frac{r'_j(x)^2}{r_j(x)}.
\]  
(E.12)

\( H_{\text{Tolhurst}}(x) \) is plotted in Figure 4 of the main paper. For \( r(x) \) of around 5 spikes or more, \( H_{\text{Tolhurst}}(x) \approx 0.5 \), and so \( \tau_{\text{Tolhurst}}(x) \) is close to \( \tilde{\tau}(x) \) (defined in Equation (11) of the main paper) with \( v = 2 \). Figure 8 of the main paper demonstrates that \( \tau_{\text{Tolhurst}}(x) \) is substantially more accurate than \( \tilde{\tau}(x) \) for very low spike rates.

### Fisher information for the Consul-Jain process

In this subsection, we derive an approximation of the Fisher information for a Consul-Jain-spiking neuron using analogous methods to the previous subsection.

Letting \( n = 0 \) in Equation (6) of the main paper, we obtain

\[
P_{\text{C-J}}(N=0 | R = r(x)) = \exp(-r(x)/\sqrt{F})
\]  
(E.13)

\[
P_{\text{Poisson}}(N = 0 | R = r(x)/\sqrt{F})
\]  
(E.14)

which has the form of \( P_{\text{Approx}} \) with \( v = \sqrt{F} \). For \( n > 0 \), we have found that, for Fano factors not too far above 1,

\[
P_{\text{C-J}}(N = n | R = r(x)) \approx u_n P_{\text{Poisson}}(N = n | R = r(x)/F),
\]  
(E.15)

which has the form of \( P_{\text{Approx}} \) with \( v = F \). Table E.2 gives the best-fitting \( u_n \) values for \( n = 1 \) to 11 when \( F = 1.5 \); Figure E.2 plots the true Consul-Jain likelihood functions for \( n = 1 \) to 11, and their approximations given by Relation (E.15).

<table>
<thead>
<tr>
<th>( n )</th>
<th>( u_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6948</td>
</tr>
<tr>
<td>2</td>
<td>0.6804</td>
</tr>
<tr>
<td>3</td>
<td>0.6757</td>
</tr>
<tr>
<td>4</td>
<td>0.6734</td>
</tr>
<tr>
<td>5</td>
<td>0.6720</td>
</tr>
<tr>
<td>6</td>
<td>0.6711</td>
</tr>
<tr>
<td>7</td>
<td>0.6704</td>
</tr>
<tr>
<td>8</td>
<td>0.6699</td>
</tr>
<tr>
<td>9</td>
<td>0.6696</td>
</tr>
<tr>
<td>10</td>
<td>0.6693</td>
</tr>
<tr>
<td>11</td>
<td>0.6690</td>
</tr>
</tbody>
</table>

Table E.2. Parameters that minimize the error in the approximation of the Consul-Jain likelihood function given by Relation (E.15) for \( F = 1.5 \). These parameters were fitted by using Equation (E.2) to evaluate the right hand side of Relation (E.15) over a range of \( r(x) \) from 0 to 100 in steps of 0.01, and comparing with the true Consul-Jain likelihood function (the left hand side of Relation (E.15), given by Equation (6) of the main paper) for the same values of \( r(x) \). We found the \( u_n \) parameters that minimized the sum of squared differences between the left and right sides of Relation (E.15) when \( F = 1.5 \). All decimal expansions are shown to an accuracy of 4 significant figures.
Figure E.2. Consul-Jain likelihood functions with $F = 1.5$, plotted against best-fitting approximations from Relation (E.15). $r(x)$ is the mean number of spikes elicited by stimulus $x$. The thin, coloured curves plot the true Consul-Jain likelihood functions (Equation (6)) with $F = 1.5$; the thick, black curves show the approximations given by the right hand side of Relation (E.15), with the $u_k$ parameter values in Table E.2.

So, for a proportion $P(N = 0)$ of trials, the Consul-Jain distribution is $P_{\text{approx}}$ with $v = \sqrt{F}$, for which the Fisher information (given by Equation (E.3)) is $r'(x)^2 / \sqrt{F r(x)}$; for the remaining proportion of trials, $[1 - P(N = 0)]$, the Consul-Jain distribution is well approximated by $P_{\text{approx}}$ with $v = F$, for which the Fisher information is $r'(x)^2 / [F r(x)]$. Since the Fisher information is a trial-averaged quantity, we can calculate it separately for these two different scenarios ($N = 0$ and $N > 0$), and then average them, weighted according to their probabilities, to give a close approximation of the Fisher information for a single, Consul-Jain-spiking neuron:

$$J \approx \left( \frac{P(N = 0)}{\sqrt{F}} + \frac{1 - P(N = 0)}{F} \right) \frac{r'(x)^2}{r(x)}.$$

Equation (E.13) shows that, for the Consul-Jain process,

$$P(N = 0) = \exp\left(-\frac{r(x)}{\sqrt{F}}\right).$$

Using Equation (E.17) to substitute for $P(N = 0)$ in Relation (E.16), we obtain

$$J \approx H_{CJ}(x) \times \frac{r'(x)^2}{r(x)}$$

where

$$H_{CJ}(x) = \frac{\exp\left(-\frac{r(x)}{\sqrt{F}}\right)}{\sqrt{F}} + \frac{1 - \exp\left(-\frac{r(x)}{\sqrt{F}}\right)}{F}$$

$$= \exp\left(-\frac{r(x)}{\sqrt{F}}\right) \left(\frac{1}{\sqrt{F}} - \frac{1}{F}\right) + \frac{1}{F}.$$
For a population of statistically independent neurons, the Fisher information is the sum of the Fisher information of the individual neurons. This gives us a good approximation of the decoding precision for a population of independent Consul-Jain-spiking neurons:

\[
\tau_{C-J}(x) = \sum_{j=1}^{K} H_{C-J}(x) \times \frac{r'_j(x)^2}{r_j(x)}. \tag{E.20}
\]

\(H_{C-J}(x)\) is plotted in Figure E.3 for several different Fano factors, \(F\). For sufficiently high \(r(x)\), \(H_{C-J}(x) \approx 1/F\), and so \(\tau_{C-J}(x)\) is close to \(\bar{r}(x)\) (defined in Equation (11) of the main paper) with \(v = F\). For Fano factors around the top of the physiologically plausible range (around 3), the mean spike count only needs to be about 7 spikes for \(\bar{r}(x)\) with \(v = F\) to be within 1% of \(\tau_{C-J}(x)\).

Figure E.4 plots the true Fisher information (thin, coloured lines) as a function of contrast for a single Consul-Jain-spiking neuron with \(F = 1.5\) (a typical value for cortical neurons), and compares this against \(\bar{r}(x)\) with \(v = F\) (thick, black lines). \(\bar{r}(x)\) provides an excellent approximation of the true Fisher information for moderate or high spike rates, but starts to get inaccurate when the spike rate gets very low. Figure E.5 plots the true Fisher information against \(\tau_{C-J}(x)\) with the same Fano factor, \(F = 1.5\).

\(\tau_{C-J}(x)\) provides an excellent match to the Fisher information at all spike rates.
The thin, coloured curves plot a single neuron's true Fisher information for decoding contrast, calculated numerically (see Supplementary Appendix H for methods). The neuron had a Consul-Jain spiking process and a Naka-Rushton tuning function with $z = -1$, $q = 3$, and $n_0 = 0$. The corresponding thick, black curves plot the approximations given by $\tilde{f}(x)$ with $v = 1.5$ (Equation (11) of the main paper).
Figure E.5. The same as Figure E.4, except that the thick, black curves plot the approximations given by $\tau_{C,J}(x)$ (Equation (E.20)).
Appendix F: Fisher information for the approximate spiking distribution

The Fisher information, $J$, for a single neuron decoding a stimulus of value $x$ is given by the average negative 2nd derivative of the log-likelihood function:

$$J = -\frac{d^2 \ln P(N=n \mid X=x)}{dx^2}, \quad (F.1)$$

where $\langle y \rangle$ is the trial-averaged value of $y$. Since the mean spike rate, $r(x)$, is a deterministic function of $x$, we have

$$P(N=n \mid X=x) = P(N=n \mid R=r(x)). \quad (F.2)$$

Substituting Equation (F.2) into (F.1), we obtain

$$J = -\frac{d^2 \ln P(N=n \mid R=r(x))}{dx^2}. \quad (F.3)$$

Suppose the neuron’s true likelihood function was given by $P_{\text{Approx}}(N=n \mid R=r(x))$, defined in Equation (E.2). Then

$$\ln P_{\text{Approx}}(N=n \mid R=r(x)) = (n/v)\ln(r(x)) - r(x)/v + (n/v)\ln(1/v) + \ln\left(\frac{u_n}{\Gamma((n/v)+1)}\right) \quad (F.4)$$

$$= \frac{1}{v^2}[n\ln(r(x)) - r(x)] + \text{terms independent of } x. \quad (F.5)$$

Note that the value of $u_n$ is irrelevant to the Fisher information, because it gets absorbed into the “terms independent of $x$”, which disappear when we differentiate.

Differentiating twice with respect to $x$, we get

$$\frac{d^2 \ln P_{\text{Approx}}(N=n \mid R=r(x))}{dx^2} = \frac{1}{v^2}\left[r''(x) - \frac{r''(x)}{r(x)}\right]. \quad (F.6)$$

The trial-averaged value of this gives us the Fisher information for this distribution:

$$J = \frac{1}{v}\left[r''(x) - \frac{r''(x)}{r(x)}\right]. \quad (F.7)$$

The step from Equation (F.7) to (F.8) follows from the fact that $\langle an+b \rangle = a\langle n \rangle + b$ for constant $a$ and $b$ (note that, for a given $x$, the only term on the right hand side of Equation (F.7) that varies across trials is $n$). Since $r(x)$ is the mean spike count, we have

$$\langle n \rangle = r(x). \quad (F.9)$$

Using Equation (F.9) to substitute for $\langle n \rangle$ in Equation (F.8), we obtain

$$J = \frac{1}{v}\frac{r'(x)^2}{r(x)}. \quad (F.10)$$

Note that, for $v=1$ and $u_n=1$ for all $n$, the “approximate” probability distribution is exactly the Poisson, so an exact expression for the Fisher information of the Poisson distribution is given by

$$J = \frac{r'(x)^2}{r(x)}. \quad (F.11)$$

(see also Dayan & Abbott, 2001, Chapter 3).

Appendix G: Estimating decoding precision

Our estimates of decoding precision were obtained using methods essentially the same as those of Clatworthy et al. (2003) and Chirimuuta et al. (2003). For each neuron, $j$, in the population, we precalculated $P(N_j=n_j \mid X=x)$ for a large range of stimulus values, $x$, and spike counts, $n_j$. For each $n_j$, $x$ was varied in discrete steps of 0.01 from −3 to 0.1 (to be compatible with Clatworthy et al.). For each $x$, $n_j$ took all integer values from 0 to a number beyond which the maximum of $P(N_j=n_j \mid X=x)$ over all $x$ was negligible. The precalculated likelihood fun-
functions were used both to generate the spikes and to decode them.

In each simulation, the precalculated values of $P(N_j = n_j \mid X = x)$ for each neuron, $j$, were given by

$$P(N_j = n_j \mid X = x) = P_{\text{Tothurst}}(N_j = n_j \mid R = r(x)).$$

We used these values to precalculate the cumulative probability distributions as well. On each trial with stimulus level $x$, each possible value of $n_j$ had a probability of occurrence given by Equation (G.1). This was achieved by generating a random number sampled from a flat distribution between 0 and 1, and then finding the lowest $n_j$ for which the cumulative probability distribution of $n_j$ was greater than or equal to the random number.

After generating the spike counts, we used the precalculated likelihood functions to decode them. On each trial, we needed to find the stimulus level, $x$, that maximized the stimulus likelihood, $P(N = n \mid X = x)$, where $N$ is a random variable representing the population response, and $n$ is its value. For statistically independent neurons, $P(N = n \mid X = x)$ is the product of $P(N_j = n_j \mid X = x)$ over the population (see Equation (8) of the main paper). For large populations, this product can be too small to represent using floating point values on a standard computer, so instead we maximized $\ln P(N = n \mid X = x)$ (which increases monotonically with $P(N = n \mid X = x)$, and therefore peaks at the same value of $x$).

$\ln P(N = n \mid X = x)$ is given by $\sum_j \ln P(N_j = n_j \mid X = x)$.

The stimulus estimate, $\hat{x}$, was the value of $x$ that maximized the likelihood. We repeated this process 10,000 times for each stimulus value, $x$, and the precision was calculated using Equation (10) of the main paper.

**Appendix H: Numerical calculation of true Fisher information**

For a single neuron, the general expression for the Fisher information, $J$, can be written as

$$J = \left\langle \left( \frac{d \ln P(N = n \mid X = x)}{dx} \right)^2 \right\rangle$$

where $\langle y \rangle$ is the trial-averaged value of $y$ (see Dayan & Abbott (2001), p. 109). Since the mean spike rate is a deterministic function of $x$, we have $P(N = n \mid X = x) = P(N = n \mid R = r(x))$. The expression for $P(N = n \mid R = r(x))$ depends on the spiking process and the tuning function, $r(x)$. To numerically calculate the Fisher information for a single neuron, we first calculated $\ln P(N = n \mid R = r(x))$ for all $x$ from $-3$ to $0.1$ in steps of $\delta x = 0.001$, and for all integers, $n$, from 0 to $M$, where $M$ was high enough for the value of $\ln P(N = n \mid R = r(x))$ to be negligible. For each $n$, we then numerically differentiated $\ln P(N = n \mid R = r(x))$ with respect to $x$, to give, for each $x$ and $n$, a close approximation of the derivative, $d \ln P(N = n \mid R = r(x))/dx$. This approximation is given by

$$\frac{\ln P(N = n \mid R = r(x + \delta x)) - \ln P(N = n \mid R = r(x - \delta x))}{2\delta x}.$$  

The numerical value, $J_{\text{numerical}}(x)$, of the Fisher information was then calculated from this numerical derivative using a discrete approximation of Equation (H.1):

$$J_{\text{numerical}}(x) = \sum_{n=0}^{M} P(N = n \mid R = r(x)) \times \frac{\left( \ln P(N = n \mid R = r(x + \delta x)) - \ln P(N = n \mid R = r(x - \delta x)) \right)^2}{2\delta x}.$$  

The weighted sum in Equation (H.3) approximates the mean value of the squared derivative across trials.

**Appendix I: Generating semi-saturation contrasts for cat and monkey populations of neurons**

Our distributions of cat and monkey $z$ values (i.e. $\log_{10}(c_{1/2})$ values) were based on the histograms in Clatworthy et al.’s (2003) Figure 6, which show the distributions of semi-saturation contrasts obtained from fitting Naka-Rushton functions to the contrast-response functions of V1 neurons in many different physiological experiments. These histograms show the number of neurons falling into each bin of width $\delta z = 0.2 \log_{10}$ contrast units. Clatworthy et al.’s Figure 6 plots the number of neurons in each bin as a function of the $z$-value corresponding to the centre of the bin, for cat and monkey populations, and these distributions are re-plotted as filled...
circles in our Figure I.1. We fitted smooth probability density functions (PDFs) to these distributions, using a maximum-likelihood fit.

For the cat data, the fitted PDF, $f_{\text{cat}}(z)$, was a single Gaussian, with two parameters, the mean, $\mu$, and the standard deviation, $\sigma$:

$$f_{\text{cat}}(z) = \frac{1}{\sigma \sqrt{2\pi}} \exp \left( -\frac{(z-\mu)^2}{2\sigma^2} \right). \quad (I.1)$$

Assuming $f_{\text{cat}}(z)$ is the true PDF, and the histogram bin width, $\delta z$, is small, the probability, $P(z)$, of a neuron falling in a bin centred on $z$ is closely approximated by $P(z) = f_{\text{cat}}(z) \delta z$. The log likelihood of the parameters $(\mu, \sigma)$ is the log probability of the data given the parameters, i.e. $\sum_z m(z) \ln P(z)$, where summation is over the bin centres, $z$, and $m(z)$ is the number of neurons in the bin centred on $z$. We found the parameters of the PDF with the highest log likelihood, and these were given by $\mu = -0.9928$ and $\sigma = 0.3833$. This PDF is plotted as the smooth curve in our Figure I.1A, scaled for the number of neurons in the population.

The Monkey data in Clatworthy et al.’s Figure 6 showed two peaks, and so the distribution was fitted with a PDF, $f_{\text{monkey}}(z)$, formed from the sum of two Gaussians:

$$f_{\text{monkey}}(z) = \frac{f_1(z) + f_2(z)}{\int_{-\infty}^{\infty} f_1(z) + f_2(z) dz}, \quad (I.2)$$

where

$$f_1(z) = A \exp \left( -\frac{(z-\mu_1)^2}{2\sigma_1^2} \right), \quad (I.3)$$

$$f_2(z) = \exp \left( -\frac{(z-\mu_2)^2}{2\sigma_2^2} \right), \quad (I.4)$$

and

$$\int_{-\infty}^{\infty} f_1(z) + f_2(z) dz = \int_{-\infty}^{\infty} f_1(z) dz + \int_{-\infty}^{\infty} f_2(z) dz = A \sigma_1 \sqrt{2\pi} + \sigma_2 \sqrt{2\pi}$$

$$= (A \sigma_1 + \sigma_2) \sqrt{2\pi} \quad (I.5)$$

The monkey PDF was fitted to Clatworthy et al.’s data using the same maximum-likelihood method as for the cat data, except that the monkey PDF had five parameters, rather than two. The fitted values were $A = 3.643$, $\mu_1 = -0.7247$, $\sigma_1 = 0.3985$, $\mu_2 = 0.6747$, and $\sigma_2 = 0.1927$. The scaled PDF is plotted as the smooth curve in our Figure I.1B.

Figure I.1. Distributions of semi-saturation contrasts in V1. (A) Cat data. (B) Monkey data. Filled circles show frequency data from Clatworthy et al.’s (2003) Figure 6. Smooth curves show the PDFs that we fitted to these data, vertically scaled for the size of the population. The vertical scaling was performed by taking the PDF, and then multiplying its height by $M \times \delta z$, where $\delta z$ is the width of each histogram bin (0.2), and $M$ is the number of neurons in Clatworthy et al.’s histogram (cats: $M = 138$ neurons; monkey: $M = 219$ neurons). This scaling meant that the function gave the expected number of neurons in each bin. Magenta vertical lines indicate the 18 semi-saturation contrasts used for each animal in Figure 7 of the main paper.
The smooth curves were truncated at \( z = 0 \) at the top end, to be compatible with Clatworthy et al, who excluded all neurons with \( c_{1/2} > 1 \) (i.e. \( z > 0 \)). At the bottom end, the monkey curve was truncated at \( z = -2 \), and the cat curve was truncated at \( z = -1.6 \). The lower limit for the monkey data corresponded to the centre of the leftmost nonzero histogram bin in Clatworthy et al.’s Figure 6B. The lower limit for the cat data corresponded to the centre of the second-to-left nonzero histogram bin in Clatworthy et al.’s cat data; the reason for this choice for the cat data was that it gave a better fit to Clatworthy et al.’s accuracy scores than setting it to the centre of the leftmost bin, although both gave a good fit. The amplitudes of these truncated PDFs were rescaled so that they integrated to 1 (making them true PDFs), and then we generated corresponding cumulative distribution functions that varied between 0 and 1. \( K \) \( z \)-values were then obtained by reading off the \( z \)-values corresponding to cumulative probabilities evenly spaced from 0 to 1, in steps of \( \frac{1}{(K - 1)} \). These \( z \)-values are shown as vertical lines in Figure I.1 for the example of \( K = 18 \), as used in Figure 7 of the main paper.

### Appendix J: Simulation methods for 2AFC contrast detection tasks

The model was set up the same as for estimating decoding precision (Appendix G) except that the set of contrasts over which the likelihood functions were precalculated included zero Michelson contrast (i.e., \( x = -\infty \)). For Naka-Rushton exponent \( q = 1 \), the likelihood functions were precalculated over values of \( x = -\infty \) and \( x = -7 \) to 0 in steps of 0.01, and the target log contrast ranged from -7 to 0 in steps of 0.05. For \( q = 2 \) to 5, the likelihood functions were precalculated over values of \( x = -\infty \) and \( x = -5 \) to 0 in steps of 0.01, and the target log contrast ranged from -5 to 0 in steps of 0.05.

\( q \) took values of 1, 2, 3, 4, or 5; \( r_{\text{max}} \) took values of 1, 2, 4, 8, 16, 32, 64, 128, 256, or 512. We simulated a 2AFC detection task with each combination of these parameters. For each combination, all the neurons had identical contrast-response functions, with \( c_{1/2} = 0.025 \).

On each 2AFC trial, we generated spikes for the given target contrast as described in Appendix G. The zero-contrast stimulus always gave zero spikes, because \( r_0 = 0 \) in these stimulations. The model responded correctly on 2AFC trials on which the target elicited at least one spike; for each contrast level, we counted up these 2AFC trials, and then added half the remaining 2AFC trials, on which the model would have had to guess (with 0.5 probability of guessing correctly). This gave the model’s total number of “correct” responses, which we divided by the total number of 2AFC trials (10,000) to give the proportion correct. For each parameterization of the model, a 3-parameter Weibull function (Equation (32) of the main paper) was fitted to the proportion of correct responses as a function of target Michelson contrast.

### Appendix K: The additive noise distribution that gives a Weibull psychometric function

Here, we prove that, assuming that the response of each detector, \( j \), is a linear function of contrast plus a sample of additive, statistically independent noise, and the observer detects the stimulus if at least one detector responds above its sensory threshold, \( \theta_j \), then the observer’s psychometric function will be a Weibull function with slope \( \beta \) when the noise on each detector has a cumulative distribution function (CDF), \( F \), given by

\[
F_{E_j}(\varepsilon) = P(E_j \leq \varepsilon) = \begin{cases} 
1 & \varepsilon > \theta_j \\
\exp\left[ -\left( \frac{\theta_j - \varepsilon}{\beta_j} \right)^\beta \right] & \varepsilon \leq \theta_j
\end{cases}
\]

(K.1)

The term \( E_j \) in Equation (K.1) is a random variable representing the noise added to detector \( j \). \( \varepsilon \) is a specific instance of \( E_j \). By assumption, the response, \( \eta_j \), of each detector, \( j \), is given by

\[
\eta_j = c_j \theta_j + E_j
\]

(K.2)

where \( \alpha_j \) is the reciprocal of that detector’s sensitivity to the stimulus. Also, by assumption, the probability that the observer detects the stimulus is the probability that not all the responses fall below threshold:

\[
P(\text{detection}) = 1 - P(\forall j \, \eta_j \leq \theta_j)
\]

(K.3)

\[
= 1 - P(\forall j \, c_j / \alpha_j + E_j \leq \theta_j)
\]

(K.4)

\[
= 1 - P(\forall j \, E_j \leq \theta_j - c_j / \alpha_j)
\]

(K.5)

\[
= 1 - \prod_{j=1}^{K} P( E_j \leq \theta_j - c_j / \alpha_j )
\]

(K.6)
where $K$ is the number of detectors.

Now, let us assume that the CDF of the noise, $F_{E_j}$, in Equation (K.7) is given by Equation (K.1). Since both contrast, $c$, and sensitivity, $1/\alpha_j$, are nonnegative values, we have $c/\alpha_j \geq 0$, and therefore $\theta_j - c/\alpha_j \leq \theta_j$.

Substituting $\theta_j - c/\alpha_j$ for $\varepsilon$ in Equation (K.1), we obtain

$$F_{E_j}(\theta_j - c/\alpha_j) = \exp\left[-\left(c/\alpha_j\right)^\beta\right].$$

(K.8)

Using Equation (K.8) to substitute for $F_{E_j}(\theta_j - c/\alpha_j)$ in Equation (K.7) gives

$$P(\text{detection}) = 1 - \prod_{j=1}^K \exp\left[-\left(c/\alpha_j\right)^\beta\right]$$

(K.9)

$$= 1 - \exp\left[-\left(c/\alpha\right)^\beta\right]$$

(K.10)

where

$$\alpha = \left(\sum_{j=1}^K \alpha_j^{-\beta}\right)^{-1/\beta}.$$  

(K.11)

Thus, the psychometric function is a Weibull function with threshold, $\alpha$, determined from the sensitivities of the individual detectors using exactly the same equation as that of Quick. For application to 2AFC experiments, Equation (K.10) must be corrected for guessing, which yields Equation (22) of the main paper. □

Note that, since sensitivity is nonnegative, the contrast threshold given by Equation (K.11) is always above zero, and so Tyler and Chen’s (2000) argument that “high-threshold probability summation fails for additive noise” does not apply when the psychometric function has the form of a Weibull function.

The noise probability density function (PDF) corresponding to the CDF in Equation (K.1) can be found by differentiating Equation (K.1) with respect to $\varepsilon$:

$$f_{E_j}(\varepsilon) = \beta(\theta_j - \varepsilon)^{\beta-1} \exp\left[-(\theta_j - \varepsilon)^\beta\right].$$

(K.12)

This function is plotted in Figure K.1 for $\beta = 1.3$, 2, 4, and 8, with $\varepsilon$ expressed in units such that the detector’s sensory threshold, $\theta_j$, is equal to 1.

Figure K.1. The Weibull noise PDF, as defined in Equation (K.12) for four values of $\beta$. $\varepsilon$ is the noise on detector $j$, which is expressed in units such that the detector’s sensory threshold, $\theta_j$, is equal to 1.