ccPlanning

ccplanning.net

TECHNICAL WHITE PAPER

From Erlang to Excel

Contact-centre queueing theory from beginner to expert — and how to rebuild every formula in a spreadsheet

John Casey

Workforce Planning Expert · ccPlanning

Companion workbook: download the live Excel model

Contents

1. Introduction

Who this paper is for

If you have ever typed a call volume into an online Erlang calculator and trusted the number it gave back, this paper is for you. The goal is to take you from that black box to a place where you understand exactly what the calculation does, why it does it, and how to reproduce every figure yourself in a spreadsheet you control.

It is written to be read in order. The early sections assume nothing beyond arithmetic and a working knowledge of a contact centre. By the end we are deriving abandonment from a birth–death Markov chain and discussing where the classical formulae stop being trustworthy. You can stop at any point and still have something useful: a solid Erlang C model, an Erlang C model with shrinkage, or the full picture including abandonment and the limits of the theory.

> Every number in this paper is reproduced in the companion workbook, `erlang-to-excel-companion.xlsx`. The worked example used throughout — 250 contacts in a 30-minute interval at 240-second handle time, targeting 80% answered in 20 seconds — appears in cell B5 of that workbook so you can change it and watch everything move.

Get the workbook: ccplanning.net/downloads/erlang-to-excel-companion.xlsx

The man behind the formulae

The mathematics in this paper carries one man's name. Agner Krarup Erlang (1878–1929) was a Danish mathematician who graduated from the University of Copenhagen in 1901 with a sweep of subjects — mathematics, physics, astronomy and chemistry — and then spent the better part of a decade as a schoolteacher, pursuing probability theory privately in his spare time. The decisive turn came in 1908, when he joined the Copenhagen Telephone Company as a scientific collaborator and head of its newly formed laboratory. For the first time a serious working mathematician was sitting inside a real, expensive operational problem.

That problem was provisioning. A telephone exchange needs circuits, and circuits cost money: install too few and callers meet the engaged tone, install too many and capital sits idle. Erlang reframed this as a question of probability rather than guesswork. His 1909 paper, "The Theory of Probabilities and Telephone Conversations", used real data from the Copenhagen exchange to show that calls arrive at random, following a Poisson distribution — the very assumption we lean on in Section 4. His 1917 paper then gave the loss and waiting-time formulae we still call Erlang B and Erlang C. From one practical need, he founded the disciplines of teletraffic engineering and queueing theory.

A century later, the same equations size far more than telephone trunks: contact-centre agents, computing clusters, checkout lanes, hospital beds — anywhere finite capacity meets random demand and service must be balanced against cost. The international unit of traffic load, the erlang, was named in his honour in 1946. So when you calculate the staffing for "80% answered in 20 seconds", you are reaching for a tool a modest Danish engineer built to stop Copenhagen's switchboards from jamming — and that is precisely why it still matters today.

2. The shape of the problem

Staffing a contact centre is a queueing problem. Work arrives at random; a finite number of agents handle it; if everyone is busy, the next contact waits. We want to know one thing above all: how many agents do we need on the phones in a given interval so that a defined proportion of contacts is answered within a defined time — the service level, conventionally written as “80% in 20 seconds”.

What makes this hard is randomness. If 250 calls arrived exactly evenly across 30 minutes, and each took exactly 240 seconds, you could solve the staffing with a single division. They do not. Calls cluster; some handle times are short and some long. It is precisely this variability that creates queues — and queueing theory is the mathematics of putting a number on it.

The classical answer is a family of formulae developed by the Danish engineer A. K. Erlang for telephone networks a century ago. Three of them matter to us: Erlang B (what happens when a busy system simply blocks new arrivals), Erlang C (what happens when they queue), and Erlang A (what happens when waiting callers lose patience and abandon). We build them up in that order.

3. Offered load: the number everything hangs on

Before any service-level calculation, we need the workload presented to the centre in a single interval. This is the offered load, measured in a dimensionless unit called the Erlang. One Erlang is one continuous hour of work per hour of clock time — equivalently, the average number of contacts in progress simultaneously if there were no queueing at all.

$$A = (\text{contacts} \times \text{AHT}) / \text{interval length}$$

All three quantities must be in the same time unit. Working in seconds, with 250 contacts in a 30-minute (1,800-second) interval at an average handle time of 240 seconds:

$$A = (250 \times 240) / 1{,}800 = 33.33 \text{ Erlangs}$$

That figure is the floor on staffing: you can never meet demand with fewer than 33.33 agents, and because agents are whole people who also need slack to keep queues from exploding, you will always need meaningfully more. Exactly how many more is what Erlang C tells us.

In Excel

With the interval in minutes in B6, contacts in B5 and AHT in seconds in B7:

```
B13:  =B6*60          (interval length in seconds)
B14:  =B5*B7/B13        (offered load A, in Erlangs)
```

4. The assumptions behind the maths

Erlang's formulae are exact — but only for a specific idealised system. Being honest about that system is what separates a practitioner from a calculator operator. The classical models assume:

- **Poisson arrivals** — contacts arrive independently and at random, so the number arriving in an interval follows a Poisson distribution (its variance equals its mean). Real arrivals are often "burstier" than this.
- **Exponential handle times** — service durations follow an exponential distribution, which is "memoryless": how long a call has already lasted tells you nothing about how long remains. Real handle times are usually closer to lognormal, with a fatter tail.
- **A single, homogeneous pool** — every agent can take every contact, and every contact is interchangeable. The moment you have skills and routing, this breaks (Section 8).
- **Statistical equilibrium** — the arrival rate is constant for long enough that the queue settles into a steady state within the interval.

None of these is perfectly true. That does not make the formulae useless — it makes them a well-understood approximation. Knowing where each assumption bites lets you decide when Erlang C is good enough and when you need to reach for simulation.

5. Erlang B: blocking, and the recursion that tames it

Erlang B answers a narrower question than ours: in a system with n servers and no waiting room, what fraction of arrivals find every server busy and are turned away ("blocked")? It is the model for trunk lines and any resource that has no queue. We need it for one practical reason: Erlang C is most stably computed from Erlang B.

The textbook form involves the offered load raised to the power n, divided by a factorial — both of which overflow a spreadsheet long before n reaches realistic contact-centre sizes. The fix is a recursion that never forms those large numbers. Writing B(n,A) for the blocking probability with n servers and load A:

$$B(0, A) = 1$$

$$B(n, A) = A \cdot B(n-1, A) / (n + A \cdot B(n-1, A))$$

Each step uses only the previous value, the server count, and the load. It is numerically stable for any n you will ever need, and it is trivial to lay down a column. As a sanity check, B(1,1) = 0.5 and B(2,2) = 0.4 — values you can verify by hand.

In Excel

Put the offered load A in B14. Start a table with n = 0 in A20 and a seed value of 1 in B20. Then in row 21 (n = 1) and down:

```
A21:  =A20+1
B21:  =$B$14*B20/(A21+$B$14*B20)      copy down
```

Column B is now Erlang B for every staffing level, computed without a single factorial.

6. Erlang C: the queueing workhorse

Erlang C is the model behind almost every staffing calculator in our industry. It assumes the same Poisson/exponential world as Erlang B, but with an unlimited queue and infinitely patient callers: nobody is blocked and nobody abandons — they wait as long as it takes.

6.1 The probability of waiting

The central quantity is the probability that an arriving contact has to wait at all (i.e. finds all n agents busy). It is written C(n,A) and follows directly from Erlang B:

$$C(n, A) = n \cdot B(n,A) / (n - A + A \cdot B(n,A))$$

This is only valid when n > A. If n ≤ A the system is unstable — work arrives faster than it can be cleared, the queue grows without bound, and the probability of waiting is effectively 1.

6.2 Service level

From the probability of waiting we get the service level: the probability that a contact is answered within a target time t. Intuitively, an arriving contact either does not wait at all, or waits in a queue that drains at the rate of the spare capacity (n − A):

$$SL(t) = 1 - C(n,A) \cdot e^{(-(n-A) \cdot t / AHT)}$$

6.3 Average speed of answer and occupancy

Two more outputs fall straight out of the same quantities. Average speed of answer (mean wait across all contacts, including those answered immediately):

$$ASA = C(n,A) \cdot AHT / (n - A)$$

And occupancy — the fraction of paid agent time spent actually handling contacts:

$$Occupancy = A / n$$

Occupancy is the quiet constraint. A roster that hits service level at 95% occupancy will burn agents out and inflate attrition; most centres cap planned occupancy around 85%. The companion workbook lets you set that cap and refuses to recommend a number that breaches it.

6.4 Finding the staffing number

There is no closed-form way to invert these equations for n. Instead we compute service level (and occupancy) for each candidate staffing level and take the smallest n that meets both targets. This is exactly what a spreadsheet table does well.

6.5 Worked example

For our running example — A = 33.33 Erlangs, target 80% in 20 seconds — the table below shows how the key metrics behave as we add agents. Notice how violently service level swings near the answer: this is the “power of one” that every real-time analyst knows in their bones.

Agents n	Erlang B	Erlang C	Service level	ASA (s)	Occupancy
34	0.1147	0.8686	17.8%	312.7	98.0%
35	0.0985	0.6965	39.4%	100.3	95.2%
36	0.0836	0.5519	55.8%	49.7	92.6%
37	0.0700	0.4318	68.2%	28.3	90.1%
38	0.0579	0.3334	77.4%	17.2	87.7%
39	0.0471	0.2540	84.2%	10.8	85.5%
40	0.0378	0.1907	89.1%	6.9	83.3%

The smallest staffing level meeting 80% in 20 seconds is **39 agents**, achieving 84.2% service level, a 10.8-second average speed of answer and 85.5% occupancy. Thirty-eight agents would reach only 77.4% — the difference of a single person is six points of service level.

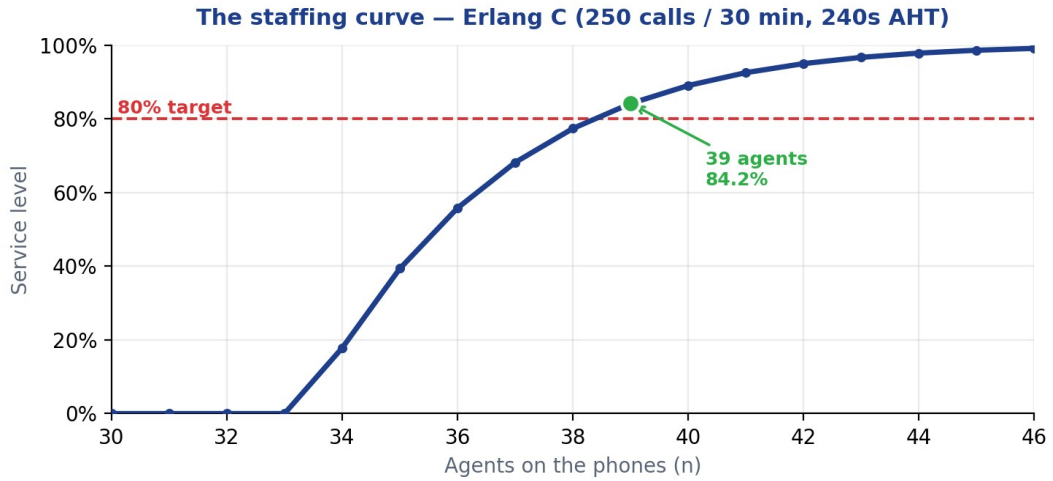


Figure 1. Service level climbs an S-curve as agents are added. The gain from one extra agent is huge near the steep middle and tiny once you are comfortably above target.

6.6 Building Erlang C in Excel

Extend the Erlang B table from Section 5 with four more columns. With A in B14, target time in B9, AHT in B7, target SL in B8 and the occupancy cap in B10:

```
C21:  =IF(A21<=$B$14,1,A21*B21/(A21-$B$14+$B$14*B21))     (Erlang C)
E21:  =IF(A21<=$B$14,0,1-C21*EXP(-(A21-$B$14)*$B$9/$B$7))  (service level)
F21:  =IF(A21<=$B$14,"",C21*$B$7/(A21-$B$14))           (ASA)
G21:  =IF(A21=0,0,$B$14/A21)                            (occupancy)
H21:  =IF(AND(E21>=$B$8,G21<=$B$10),"yes","")           (meets target?)
I21:  =IF(H21="yes",A21,"")                             (n if it qualifies)
```

The recommended staffing number is then the smallest qualifying n, with its metrics pulled back by lookup:

```
Agents required:  =MIN(I20:I100)
Achieved SL:      =INDEX(E20:E100,MATCH(<agents>,A20:A100,0))
```

That single MIN/INDEX pairing replaces the “iterate until it passes” loop that a calculator runs internally — and you can see every step.

7. Erlang A: adding human patience

Erlang C’s biggest fiction is infinite patience. Real callers abandon, and abandonment is not a rounding error — it both relieves the queue (an abandoned call stops consuming a future agent) and represents lost service. Erlang A, also called the Palm or M/M/n+M model, adds a patience parameter and lets us estimate abandonment directly.

7.1 The model

We model the system as a birth–death process: the “state” is the number of contacts in the system (in service plus waiting), arrivals push the state up, and completions or abandonments pull it down. Three rates govern it: the arrival rate λ = contacts⁄interval, the service rate per agent μ = 1⁄AHT, and the abandonment rate per waiting caller θ = 1⁄(average patience).

7.2 State probabilities

We build the relative probability (an unnormalised “weight”) of each state k from the one below it. Up to n contacts, only agents are clearing work; beyond n, the waiting callers also abandon at rate θ each:

$$w(0) = 1$$

$$w(k) = w(k-1) \cdot A/k \qquad \text{for } k \le n$$

$$w(k) = w(k-1) \cdot \lambda/(n\cdot\mu + (k-n)\cdot\theta) \qquad \text{for } k > n$$

Dividing each weight by the sum of all weights turns them into genuine probabilities p(k). The state space is infinite in theory; in practice we truncate at a comfortably large k (the workbook uses 300), because the probabilities vanish well before then.

7.3 Abandonment and waiting

Two results come straight from the state probabilities. The expected number of callers waiting is the sum of (k − n) · p(k) over the queued states; abandonment per arrival is that, scaled by the patience and arrival rates:

$$\text{Abandon\%} = \theta \cdot \Sigma\,(k-n)\cdot p(k) \;/\; \lambda \qquad (\text{sum over } k > n)$$

$$P(\text{wait}) = \Sigma\, p(k) \qquad (\text{sum over } k \ge n)$$

For the running example with 38 agents and 90-second average patience, this gives about 3.1% abandonment and an 18.6% chance of waiting — a more honest picture than Erlang C, which by construction can say nothing about abandonment at all.

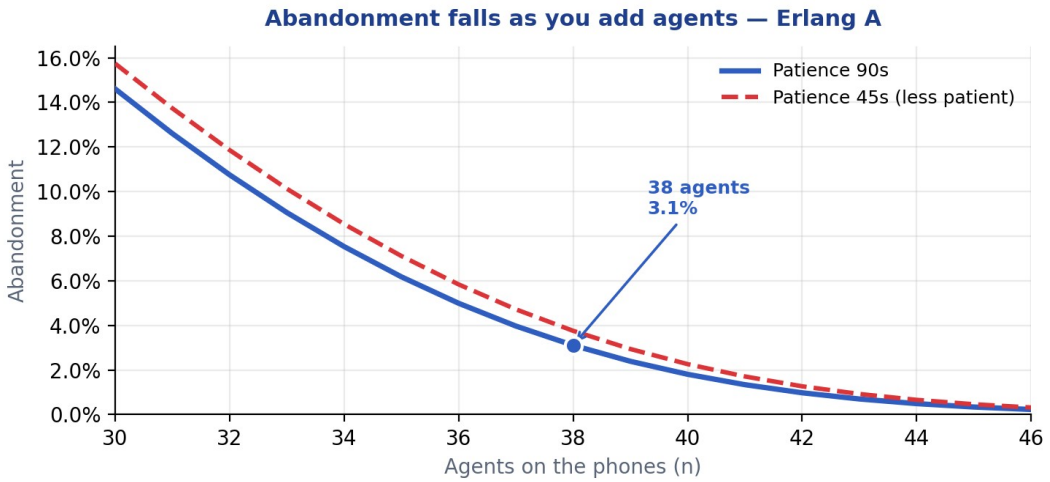


Figure 2. Abandonment against staffing for two patience levels. Less patient callers (dashed) abandon more at every staffing level, so patience — not just volume — drives the requirement.

7.4 Building Erlang A in Excel

On a fresh sheet, bring across contacts, AHT and interval seconds, then add patience (B8) and agents on duty (B9). Compute the rates, then lay down a state table with k = 0 in D21 and a seed weight of 1 in E21:

```
B13: =B5/B7   (lambda)   B14: =1/B6   (mu)   B15: =1/B8   (theta)
E22:  =IF(D22<=$B$9, E21*$B$12/D22,
         E21*$B$13/($B$9*$B$14+(D22-$B$9)*$B$15))     copy down to k=300
F22:  =E22/$B$16       (p(k); B16 = SUM of the weight column)
G22:  =IF(D22>$B$9,(D22-$B$9)*F22,0)                  (queue contribution)
Abandon%:  =B15*SUM(G:G)/B13     P(wait): =SUMIF(D:D,">="&B9,F:F)
```

> A caution: the weights can grow very large before they shrink, so for very high loads the unnormalised column can strain a spreadsheet's precision. For interval-level contact-centre sizes it is perfectly well-behaved; for an enterprise-wide roll-up, scale the load down or move to a tool built for it.

8. Erlang X: abandonment, blocking and redials

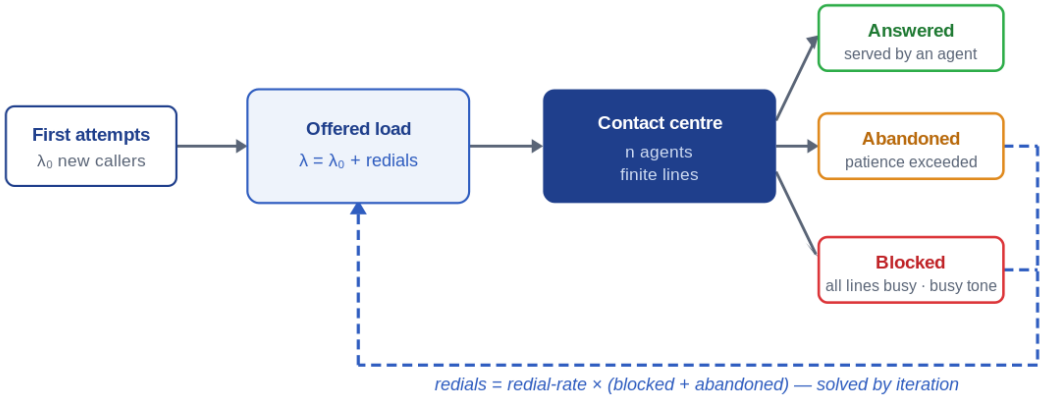
Erlang A added patience. Erlang X goes one step further and folds three real-world behaviours into a single model. It is the most realistic of the classical (non-simulation) formulae, and the one several modern workforce-management tools now use as their default.

It was assembled in the 1990s by the Dutch mathematician Ger Koole, who combined three properties that earlier models handled only in isolation:

- **Abandonment** — callers leave the queue when their patience runs out. This is the Erlang A behaviour, first brought into queueing theory by the Swedish mathematician Conny Palm in the 1960s.
- **Blocking** — a finite number of lines, or a maximum queue size, beyond which a new caller meets a busy tone or is diverted. This is the Erlang B behaviour.

- **Redials (retrials)** — and this is the genuinely new part: a fraction of the callers who abandon, or who hit a busy line, simply call back. Those redials add to the load.

The redial property is what makes Erlang X both more realistic and harder to compute, because it is circular: abandoned and blocked calls generate redials, redials add to the offered load, more load means more abandonment and blocking, which generates more redials. There is no neat closed form. The model is solved by fixed-point iteration — start from the base offered load, compute abandonment and blocking, add the resulting redials back into the load, and repeat until the numbers stop moving.



The Erlang X loop. Only answered calls leave the system cleanly; a share of the abandoned and blocked callers redial and re-enter the offered load — the circular dependency that forces an iterative solution.

8.1 Why it matters

Erlang C assumes everyone waits forever, which over-staffs an impatient queue. The “restaurant table effect” captures it: book 50 tables for a 7pm sitting, 70 couples arrive, and Erlang C has them all wait indefinitely — when in reality some couples take one look at the queue and leave, and a few drift back later. Erlang X models that natural attrition and the persistent redials, so it sizes an impatient, line-limited operation more accurately. Usually that means fewer agents than Erlang C — though not always. If your callers are patient and rarely abandon, the two barely differ.

8.2 When to reach for it

When average waits regularly creep above about 90 seconds; when a meaningful fraction of callers abandon; when lines are genuinely capped; or when you want to see the extra load that a busy signal creates through redials. For a patient, uncapped queue, Erlang A — or even Erlang C — is enough. As with every model, measure your own abandonment and redial behaviour before trusting the defaults: you can only observe the patience of the callers who actually abandoned, which tends to understate the patience of everyone who stayed.

8.3 In Excel

Erlang X is the one model in this paper you cannot lay down as a single straight column, because the redial loop is circular: the offered load depends on the blocking and abandonment it itself produces. There are two ways to resolve it. The robust one — and the one the companion workbook uses — is to wrap the Erlang A calculation in a short chain of explicit iteration passes, re-feeding the redial volume into the load until it settles, with no circular reference at all. The alternative is to enable Excel's iterative calculation (File → Options → Formulas → Enable iterative calculation) and let a single circular formula converge on itself. Either way, the building blocks are exactly the Erlang A state table and the Erlang B recursion you built in the previous sections — Erlang X is those models, iterated.

> The companion workbook does exactly this: alongside the Erlang B, C and A sheets it includes a live Erlang X sheet that solves the redial loop with twelve explicit iteration passes — no circular reference, so it recalculates correctly in Excel, LibreOffice or Google Sheets without enabling any special setting. Change patience, the number of lines and the redial rate and watch blocking, abandonment and the redial load settle. There is also a free Erlang X calculator at ccplanning.net/calculators, and several vendors (CCmath, NICE, Cinareo and others) sell ready-made Erlang X tools.

9. Where the formulae break

Reproducing the maths is one kind of mastery; knowing when not to trust it is the deeper one. Every assumption from Section 4 is a fault line.

9.1 Non-Poisson arrivals

Marketing sends an email and 400 people call in ten minutes; an outage spikes the queue. When arrivals are burstier than Poisson, real service level is worse than Erlang C predicts, because the model understates the clustering. Conversely, appointment-booked or strongly scheduled work can be smoother than Poisson, and Erlang C is then conservative.

9.2 The handle-time distribution

Erlang assumes exponential handle times. Real distributions are typically lognormal — most calls near a central value, with a long tail of complex ones. For service-level sizing this matters less than people fear (the mean does most of the work), but it matters more for the variance of wait times and for abandonment.

9.3 The interval-boundary problem

Erlang assumes the system reaches equilibrium within the interval. A 30-minute interval with a sharp intraday ramp never settles; work — and queues — spill across boundaries. Calculating each interval in isolation ignores that carry-over, which is why busy-period service level is usually worse than an interval-by-interval Erlang sum suggests.

9.4 Multi-skill: no closed form

The single biggest limit. The moment agents have overlapping skills and contacts are routed between them, there is no Erlang formula — the interactions between queues and skill groups have no clean closed-form solution. Pooled, cross-skilled operations behave better than the sum of their silos (the pooling gain), and no amount of per-queue Erlang will capture it. This is the territory of simulation.

9.5 When to simulate

A discrete-event simulation drops all four assumptions: it can use any arrival pattern and handle-time distribution, model a full skill matrix and routing rules, run the clock continuously across interval boundaries, and include patience. The cost is that it returns a distribution rather than a single exact figure, and needs many runs to converge. The practical rule: Erlang C for single-skill service-level sizing, Erlang A when abandonment is material, Erlang X when redials and capped lines also bite, and simulation for multi-skill, blended, or strongly non-stationary work.

10. Which model, when

Situation	Use	Why
Single queue, patient callers, SL target	Erlang C	The standard; fast and good enough
Single queue, callers abandon	Erlang A	Captures abandonment and queue relief
Abandonment + redials + capped lines	Erlang X	Adds redials and blocking on top of Erlang A
No-queue / trunk sizing	Erlang B	Blocking, not waiting
Multi-skill or blended routing	Simulation	No closed form exists
Very bursty or scheduled arrivals	Simulation	Poisson assumption fails
Long-range FTE / capacity	Workload + occupancy	Interval Erlang rolled up

All of these except long-range capacity are available as free, browser-based tools at **ccplanning.net** — including the multi-skill simulator that picks up exactly where this paper's formulae leave off.

Appendix A. Notation

Symbol	Meaning
A	Offered load, in Erlangs = contacts × AHT ⁄ interval
n	Number of agents (servers)
AHT	Average handle time (talk + after-call work), seconds
λ (lambda)	Arrival rate = contacts ⁄ interval seconds
μ (mu)	Service rate per agent = 1 ⁄ AHT
θ (theta)	Abandonment rate per waiting caller = 1 ⁄ patience
B(n,A)	Erlang B — probability all n servers busy (blocking)
C(n,A)	Erlang C — probability an arrival must wait
t	Service-level target time (e.g. 20 seconds)
ρ = A ⁄ n	Occupancy / utilisation

Appendix B. Every Excel formula in one place

Erlang C sheet

```
B13   =B6*60                                          interval seconds
B14   =B5*B7/B13                                      offered load A
A21   =A20+1                                          agent count (seed A20=0)
B21   =$B$14*B20/(A21+$B$14*B20)                      Erlang B (seed B20=1)
C21   =IF(A21<=$B$14,1,A21*B21/(A21-$B$14+$B$14*B21))      Erlang C
E21   =IF(A21<=$B$14,0,1-C21*EXP(-(A21-$B$14)*$B$9/$B$7))  service level
F21   =IF(A21<=$B$14,"",C21*$B$7/(A21-$B$14))              ASA
G21   =IF(A21=0,0,$B$14/A21)                          occupancy
I21   =IF(AND(E21>=$B$8,G21<=$B$10),A21,"")           qualifying n
—     =MIN(I20:I100)                                  agents required
```

Erlang A sheet

```
B13   =B5/B7                                          lambda
B14   =1/B6                                           mu
B15   =1/B8                                           theta
E22   =IF(D22<=$B$9,E21*$B$12/D22,E21*$B$13/($B$9*$B$14+(D22-$B$9)*$B$15)) state
weight
F22   =E22/$B$16                                      p(k)
G22   =IF(D22>$B$9,(D22-$B$9)*F22,0)                  queue term
—     =B15*SUM(Gcol)/B13                              abandon %
—     =SUMIF(Dcol,">="&B9,Fcol)                       P(wait)
```

Appendix C. Validating your spreadsheet

Before you trust a model, check it against values you can verify independently:

- Erlang B sanity: with A = 1, B(1,A) must equal 0.5; with A = 2, B(2,A) must equal 0.4.
- Erlang C stability: for any n ≤ A, your sheet should return probability-of-waiting 1 and zero service level — never an error.
- The worked example: A = 33.33 Erlangs should require 39 agents for 80% in 20 seconds, giving 84.2% SL, 10.8-second ASA and 85.5% occupancy.
- Erlang A reduces to Erlang C: set patience very high (say 100,000 seconds) and abandonment should fall towards zero while probability-of-waiting matches the Erlang C value.
- Cross-check against the free calculators at ccplanning.net — they use exactly the recursions in this paper.

Build it once, validate it, and you will never have to take a black-box calculator's word again.