LECTURE 16: CAMB

GONG-BO ZHAO

Learning goals

After this lecture, students should be able to:

- explain why a Boltzmann code is needed to connect primordial perturbations to observables such as C_ℓ and $P(k)$;
- describe the high-level architecture of CAMB, including which modules are responsible for input, evolution, and output;
- understand the relation between CAMB variables and the standard synchronous-gauge equations of Ma and Bertschinger [1];
- explain the roles of initial conditions, tight coupling, Boltzmann hierarchies, and line-of-sight integration;
- run a minimal CAMB calculation for CMB spectra and matter power spectra, and know where to look first when adapting the code for research.

1. Why CAMB?

CAMB, the *Code for Anisotropies in the Microwave Background*, is a public Einstein-Boltzmann solver originally developed by Lewis, Challinor, and Lasenby [3]. In current official documentation, CAMB is described as a Python and Fortran code for computing CMB spectra, CMB lensing, source-count and 21 cm angular power spectra, transfer functions, matter power spectra, and background cosmological functions [9, 8, 11]. The original slide deck emphasized three reasons students should care about it: it is efficient, reasonably modular, and widely used in cosmological parameter inference. That remains an excellent summary.

At a conceptual level, CAMB solves a simple but nontrivial problem. Cosmological data are not directly predictions of the primordial curvature spectrum. Between the primordial spectrum and the observables lie the background expansion, recombination, reionization, the coupled evolution of perturbations in photons, baryons, cold dark matter, neutrinos, and dark energy, and finally the geometric projection from three-dimensional Fourier modes to the angular or spatial spectra that we actually measure. CAMB automates this chain accurately and quickly.

The most important physical point for students is that CAMB is not a black box that creates C_ℓ or $P(k)$ by magic. It is a carefully organized implementation of linear perturbation theory plus a few optional extra ingredients such as non-linear matter corrections and CMB lensing. Once that is understood, the code becomes much less intimidating.

2. What problem is CAMB solving?

The starting point is the primordial curvature spectrum,

$$\mathcal{P}_{\mathcal{R}}(k) = A_s \left(\frac{k}{k_*}\right)^{n_s-1+\cdots}, \tag{1}$$

which inflationary models attempt to predict. A Boltzmann code turns this into transfer functions and power spectra.

For angular observables one can write schematically

$$C_\ell^{XY} = 4\pi \int \mathrm{d}\ln k\, \mathcal{P}_{\mathcal{R}}(k)\, I_\ell^X(k) I_\ell^Y(k), \tag{2}$$

where the transfer functions are line-of-sight integrals,

$$I_\ell^X(k) = \int_0^{\tau_0} \mathrm{d}\tau\; S_X(k,\tau)\, j_\ell\big[k(\tau_0-\tau)\big]. \tag{3}$$

Here S_X is a source term, j_ℓ is a spherical Bessel function, and X, Y may denote temperature, E-polarization, lensing, or more general source-window observables [2, 3, 11].

For the late-time matter field, the analogous result is

$$P_m(k,z) = \frac{2\pi^2}{k^3}\, \mathcal{P}_{\mathcal{R}}(k)\, T_m^2(k,z), \tag{4}$$

where $T_m(k,z)$ is the matter transfer function. Equation (4) is the real reason CAMB appears throughout cosmology: a single code simultaneously predicts CMB anisotropies, linear matter clustering, background distances, lensing, and many cross-observables.

The slides summarized the workflow as

initial conditions → solve the coupled differential equations → output statistics.

That is the right conceptual picture. A large fraction of the code exists only to make those three steps numerically stable and efficient.

3. The overall architecture of CAMB

The call-graph slides look busy, but the logic is simple. The official readme and current generated Fortran documentation show the same basic three-block structure emphasized in the slides: preparation, core evolution, and output [9, 10].

From "messy" graph to conceptual map. A useful student-level summary is:

1. Read and validate the cosmological parameters and numerical settings.
2. Build the background cosmology, recombination and reionization histories, primordial power spectrum, k sampling, and interpolation tables.
3. For each Fourier mode k, set initial conditions and evolve the perturbation variables in time.
4. Turn the time-dependent perturbations into source functions and then into transfer functions using spherical Bessel functions.
5. Integrate over k to obtain C_ℓ values, and separately assemble transfer functions and matter power spectra.

This is why the big dependency diagrams in the slides are not actually chaos. The code is long because cosmology is detailed, not because the underlying logic is obscure.

Main files and what they do. The most useful first-pass map of the code is the following.

File or module	Main role
`inidriver.f90`, `camb.py`	Read input parameters and construct the main parameter object; command-line or Python front end [9, 8].
`model.f90`	Defines the main input structure `CAMBparams`; this is where one adds new top-level parameters [12].
`results.f90`	Stores derived quantities and output containers such as `CAMBdata`, transfer data, and interpolation tables [9, 11].
`cmbmain.f90`	The main driver for the perturbation calculation: initialize, evolve, project, and collect results.
`equations.f90`	Background and perturbation equations, source terms, and the CMB source construction; the `output` subroutine lives here [9, 14].
`InitialPower.f90`	Initial power spectrum classes and primordial-spectrum definitions [9].
`bessels.f90`	Spherical and hyperspherical Bessel functions for the line-of-sight projection [9].
`reionization.f90`, `recfast.f90`	Ionization history and recombination/reionization models [9].
`lensing.f90`	Computes lensed CMB spectra from unlensed spectra and the lensing potential [9, 6].
`halofit.f90`	Non-linear matter-power corrections [9].
`SourceWindows` and related modules	Modern angular source windows and cross-spectra beyond the primary CMB [10, 12].

For a modern student workflow, the official FORD-generated documentation is now usually more helpful than browsing raw Fortran files one by one [10]. The slide suggestion to inspect the code remains good advice, but today the built-in documentation pages make that much easier.

4. Background evolution and gauge conventions

Background. Before evolving perturbations, CAMB must know the homogeneous background. In cosmic time,

$$H^2 = \left(\frac{\dot{a}}{a}\right)^2 = \frac{8\pi G}{3}\bar{\rho} - \frac{K}{a^2}, \qquad \dot{H} = -4\pi G(\bar{\rho} + \bar{P}) + \frac{K}{a^2}. \tag{5}$$

In conformal time τ, the code frequently uses

$$\mathcal{H} \equiv \frac{a'}{a}, \tag{6}$$

which appears in CAMB as `adotoa` in the slide notation. The official readme notes that the subroutine returning $\mathrm{d}t/\mathrm{d}a$ is used whenever the background is needed, so modifications to the background sector naturally propagate through the perturbation evolution [9].

Synchronous-gauge variables. The slides connect CAMB to Ma and Bertschinger's classic synchronous-gauge equations [1]. In that gauge the scalar metric perturbations are conventionally written as

$$ds^2 = a^2(\tau)\left[-\mathrm{d}\tau^2 + \left(\delta_{ij} + h_{ij}\right)\mathrm{d}x^i\mathrm{d}x^j\right], \tag{7}$$

with

$$h_{ij}(\mathbf{k},\tau) = \hat{k}_i\hat{k}_j h(\mathbf{k},\tau) + 6\left(\hat{k}_i\hat{k}_j - \frac{1}{3}\delta_{ij}\right)\eta(\mathbf{k},\tau). \tag{8}$$

The Einstein equations then include the well-known constraint and evolution equations

$$k^2\eta - \frac{1}{2}\mathcal{H}h' = 4\pi G a^2 \delta T^0{}_0, \tag{9}$$

$$k^2\eta' = 4\pi G a^2(\bar{\rho} + \bar{P})\theta, \tag{10}$$

$$h'' + 2\mathcal{H}h' - 2k^2\eta = -8\pi G a^2 \delta T^i{}_i, \tag{11}$$

$$h'' + 6\eta'' + 2\mathcal{H}(h' + 6\eta') - 2k^2\eta = -24\pi G a^2(\bar{\rho} + \bar{P})\sigma. \tag{12}$$

CAMB does not always expose these variables in exactly this form. The theory page explains that the equations in `equations.f90` are written using variables derived from covariant quantities, in a zero-acceleration frame equivalent to the synchronous gauge [14]. This is why the code can look unfamiliar even when it is solving the same physics.

A useful dictionary between code and equations. The slide deck gives a very helpful translation table. A compact version is:

CAMB variable	**Meaning**
`tau`	conformal time τ
`adotoa`	$\mathcal{H} = a'/a$
`grho`	$8\pi G a^2 \bar{\rho}$
`dgrho`	$8\pi G a^2 \sum_i \bar{\rho}_i \, \delta_i$
`dgq`	$8\pi G a^2 \sum_i (\bar{\rho}_i + \bar{p}_i) v_i$
`etak`	$k\eta$
`z`	$h'/(2k)$
`sigma`	$(h' + 6\eta')/(2k)$
`clxc`, `clxb`, `clxg`, `clxr`	δ_c, δ_b, δ_γ, δ_ν

This dictionary is more than notation. It tells you which variables are primary objects in the code and which are reconstructed from constraints.

5. Initial conditions

The old slides make exactly the right pedagogical point: start with initial conditions, then follow the time evolution. In CAMB, the initial conditions for scalar modes are derived

from regular super-horizon series solutions. For the adiabatic growing mode, one has the familiar synchronous-gauge relations on very large scales,

$$\delta_c = \delta_b = \frac{3}{4}\delta_\gamma = \frac{3}{4}\delta_\nu \propto (k\tau)^2, \qquad \theta_c = 0, \tag{13}$$

up to higher-order corrections [1, 14].

The slide comparing CAMB notes to the code is worth retaining because it immediately shows students how an analytic series becomes actual source code. In flat models, the implementation begins schematically as

```
x = k*tau
x2 = x*x
initv(1,i_clxg) = -chi*EV%Kf(1)/3*x2*(1-omtau/5)
initv(1,i_clxr) = initv(1,i_clxg)
initv(1,i_clxb) = 0.75_dl*initv(1,i_clxg)
initv(1,i_clxc) = initv(1,i_clxb)
```

There are two immediate lessons here. First, the code works with a regular expansion in $x = k\tau$, exactly as one expects from super-horizon perturbation theory. Second, the factor `0.75_dl` is simply the statement $\delta_b = 3\delta_\gamma/4$ for adiabatic initial conditions.

For students, this is a very good moment to emphasize that initial conditions in a Boltzmann code are not guessed numerically. They are derived analytically, expanded consistently, and then fed into the differential equation solver at an early but finite time. This is one reason the results are so stable.

6. Coupled differential equations

The central numerical task is to evolve the coupled perturbation equations. The slides highlight an important structural point: not every equation in the system is a differential equation. Some are constraints used to reconstruct variables algebraically.

Differential versus algebraic equations. A particularly clean example in the slide deck is the cold-dark-matter sector. In CAMB notation,

$$\texttt{clxcdot} = -k\,\texttt{z}, \tag{14}$$

which corresponds directly to the synchronous-gauge relation

$$\delta_c' = -\frac{1}{2}h'. \tag{15}$$

Since z = h'/(2k), Eq. (14) is exactly the same statement.

By contrast, two other quantities are reconstructed using constraints:

$$\texttt{z} = \frac{0.5\,\texttt{dgrho}/k + \texttt{etak}}{\texttt{adotoa}}, \qquad \texttt{sigma} = \texttt{z} + \frac{3}{2}\frac{\texttt{dgq}}{k^2}. \tag{16}$$

These are the code versions of Einstein constraint equations such as Eqs. (9) and (10). This is a central coding strategy: evolve a minimal, stable set of variables and recover the rest from algebraic relations.

Baryons and photons. The baryon and photon equations shown in the slides are perhaps the easiest place to see real cosmological physics in the code. In standard notation,

$$\delta_b' = -\theta_b - \frac{1}{2}h', \tag{17}$$

$$\theta_b' = -\mathcal{H}\theta_b + c_s^2 k^2 \delta_b + \frac{4\bar{\rho}_\gamma}{3\bar{\rho}_b}\,\dot{\kappa}\,(\theta_\gamma - \theta_b), \tag{18}$$

while the photon continuity and Euler equations are

$$\delta_\gamma' = -\frac{4}{3}\theta_\gamma - \frac{2}{3}h', \tag{19}$$

$$\theta_\gamma' = k^2\left(\frac{1}{4}\delta_\gamma - \sigma_\gamma\right) + \dot{\kappa}(\theta_b - \theta_\gamma). \tag{20}$$

Here $\dot{\kappa} = an_e\sigma_T$ is the Thomson scattering rate and σ_γ is the photon shear.

The corresponding slide-level code form is concise:

```
clxbdot = -k*(z + vb)
vbdot   = -adotoa*vb + cs2*k*clxb - photbar*opacity*(4._dl/3*vb - qg)

clxgdot = -k*(4._dl/3._dl*z + qg)
qgdot   = k*(clxg/4._dl - pig/2._dl) + opacity*slip
```

The code notation is different, but the physics is familiar: gravity pushes, pressure resists, and Thomson scattering ties photons and baryons together.

Tight coupling. Before recombination, the Thomson rate is so large that photons and baryons form an almost single fluid. In that regime,

$$\theta_b \simeq \theta_\gamma, \tag{21}$$

and the full system becomes stiff if treated naively. CAMB therefore uses tight-coupling approximations to replace the exact photon-baryon equations with more stable reduced equations when $|\dot{\kappa}|$ is large. Students do not need every detail on first exposure; the main lesson is that good numerical cosmology requires identifying fast and slow variables and exploiting the relevant approximations at the right time.

Higher moments and polarization. Beyond the monopole and dipole, photons and neutrinos require a Boltzmann hierarchy. Schematically, for $\ell \geq 3$ one has

$$F_\ell' = \frac{k}{2\ell+1}\left[\ell F_{\ell-1} - (\ell+1)F_{\ell+1}\right] - \dot{\kappa}F_\ell, \tag{22}$$

with analogous equations for polarization moments. The quadrupole equation couples to polarization and to metric sources, exactly as indicated on the "higher moments" slide. The lesson for students is that the acoustic physics cannot be captured by a single fluid alone: the free-streaming tails of the photon and neutrino distribution functions matter, especially for polarization and phase shifts.

7. Line-of-sight integration and the CMB source term

The great speedup of modern Boltzmann codes comes from the line-of-sight method of Seljak and Zaldarriaga [2], implemented in CAMB and extended efficiently by Lewis, Challinor, and Lasenby [3]. Rather than evolving every multipole all the way to today for every ℓ, CAMB evolves time-dependent source functions and then projects them with precomputed Bessel functions.

This is the logic behind Eqs. (2) and (3). The costly, rapidly oscillating part is the Bessel function j_ℓ, which depends only on geometry. The cosmology dependence is packaged into the much smoother source function $S_X(k,\tau)$. This separation is precisely why line-of-sight methods are fast.

What is in the temperature source? The exact CAMB source expression in `equations.f90` is long, as the slide on the `output` subroutine makes very clear. But physically it is only a sum of familiar effects. Schematic temperature-source terms include

$$S_T \sim g(\tau)\left[\Theta_0 + \Psi + \frac{1}{4}\Pi\right] + \frac{\mathrm{d}}{\mathrm{d}\tau}\left[g(\tau)\frac{v_b}{k}\right] + e^{-\kappa}(\Psi' - \Phi') + \cdots , \tag{23}$$

where $g(\tau) = \dot{\kappa} e^{-\kappa}$ is the visibility function. The three most important pieces are:

- the intrinsic plus Sachs-Wolfe source at last scattering;
- the Doppler source from the baryon velocity;
- the integrated Sachs-Wolfe source from time-varying gravitational potentials.

Polarization is generated by the local quadrupole at last scattering and reionization. The code expression looks formidable only because it is written in a numerically convenient form and includes many small bookkeeping terms.

The slide deck also hints at a wonderful pedagogical exercise: one can edit individual terms in the source expression and immediately see which physical contribution they control. That is one of the fastest ways to demystify a Boltzmann code.

8. Matter transfer functions and the 3D matter power spectrum

CAMB does not stop at the CMB. The matter transfer functions and matter power spectrum are equally important outputs. In the simplest linear description,

$$\delta_m(k,z) = T_m(k,z)\,\mathcal{R}(k), \tag{24}$$

so that Eq. (4) follows immediately. It is also useful to define the dimensionless matter power

$$\Delta^2_m(k,z) = \frac{k^3}{2\pi^2} P_m(k,z). \tag{25}$$

From this one obtains quantities such as

$$\sigma^2_R(z) = \int \frac{\mathrm{d}k}{k}\,\Delta^2_m(k,z)\,W^2(kR), \tag{26}$$

with $\sigma_8 = \sigma_{R=8h^{-1}\mathrm{Mpc}}(0)$.

The current CAMB documentation makes it straightforward to access not only total matter but also transfer variables for CDM, baryons, photons, neutrinos, and total matter excluding neutrinos [11]. This is one reason CAMB is so useful pedagogically: students can see how each species contributes, rather than looking only at the final total spectrum.

Linear and non-linear options. The original slides focused on the linear transfer pipeline. Modern CAMB also includes non-linear matter-power prescriptions through the non-linear module [9]. For lecture purposes, the right lesson is simple: always understand the linear result first. Non-linear corrections are valuable, but they are not a substitute for understanding the transfer physics that produces the linear spectrum.

Modern extensions beyond the original slide deck. The current official documentation emphasizes that CAMB can also compute angular source-window spectra, their cross-spectra, and dark-age 21 cm observables [8, 12, 10]. Conceptually, these are not a new kind of calculation. They use the same transfer-function machinery as the CMB:

$$C_L^{ij} = 4\pi \int \mathrm{d}\ln k\, \mathcal{P}_{\mathcal{R}}(k)\, \Delta_L^i(k) \Delta_L^j(k). \tag{27}$$

This is worth telling students because it connects CAMB directly to later large-scale-structure topics: once one understands transfer functions and projection kernels, many "different" observables are really variations on the same theme.

9. A practical modern CAMB workflow

The original slide deck treated CAMB primarily as a Fortran program. That viewpoint is still valuable because the numerical core is in Fortran, but the modern standard entry point is usually Python [8, 11]. The official repository documents a standard installation with

```
pip install camb
```

and notes that source installation requires a Fortran compiler such as `gfortran` [8].

Minimal Python example. A compact workflow for CMB and matter power is:

```
import camb

pars = camb.CAMBparams()
pars.set_cosmology(H0=67.4, ombh2=0.0224, omch2=0.120, mnu=0.06, omk=0.0)
pars.InitPower.set_params(As=2.1e-9, ns=0.965)
pars.set_for_lmax(2500, lens_potential_accuracy=1)
pars.WantTransfer = True
pars.set_matter_power(redshifts=[0.0, 0.5, 1.0], kmax=2.0)

results = camb.get_results(pars)
cls = results.get_cmb_power_spectra(pars, CMB_unit='muK')
kh, z, pk = results.get_matter_power_spectrum(minkh=1e-4, maxkh=1.0, npoints=200)
```

The key idea is that the same `CAMBparams` object determines both the CMB and matter calculations. The current documentation also provides background-only calls, transfer-only calls, and interpolators for $P(k, z)$ [11].

Common student tasks and where to look.

Task	Useful method or setting
Background distances and derived parameters	`camb.get_background()` or `results` background methods [11].
CMB C_ℓ values	`camb.get_results()` and `get_cmb_power_spectra()` [11].
Linear matter power	`set_matter_power()` and `get_matter_power_spectrum()` [11].
Interpolated $P(k, z)$	`get_matter_power_interpolator()` [11].
Transfer variables by species	`get_matter_transfer_data()` [11].
Command-line run from an ini file	`camb inifiles/planck_2018.ini` [8].

Good in-class parameter demos. CAMB is especially effective in lecture when one varies one parameter at a time. Three good examples are:

(1) Increase $\omega_b = \Omega_b h^2$: odd acoustic peaks grow relative to even peaks because baryon loading deepens compression phases.
(2) Increase $\sum m_\nu$: small-scale matter power is suppressed because massive neutrinos free-stream.
(3) Increase $\tau_{\rm reio}$: low-ℓ polarization is enhanced because reionization produces an additional large-angle scattering surface.

These examples show students that CAMB is not just for parameter fitting. It is also a laboratory for physical intuition.

10. How to modify CAMB for research

One of the explicit goals of the original slide deck was to help students change CAMB for their own work. That is exactly the right instinct. In practice, most modifications fall into one of four categories.

1. Add a new parameter. The current Python documentation is very explicit: to add a new parameter, add it to the `CAMBparams` type in `model.f90`, then add the corresponding field to the Python wrapper in `model.py` [12]. This is usually the first step when extending the code.

2. Modify a physical sector. If one changes the background expansion or dark-energy model, the relevant changes live in the background and dark-energy modules. If one changes recombination or reionization, the natural targets are `recfast.f90` or `reionization.f90`. If one changes initial conditions or primordial spectra, `InitialPower.f90` and the initial-condition logic are the natural places to start [9, 11].

3. Modify sources or transfer functions. If the observable is new rather than the cosmology itself, one often edits the source construction rather than the background equations. The CAMB readme explicitly points out that the `output` subroutine computes the scalar sources, and that editing the lensing-source equation can be used to compute power spectra for other tracers [9]. That is exactly the logic behind many practical extensions.

4. Validate relentlessly. After every modification, recover a baseline LCDM result, compare to the unmodified code, and vary accuracy settings. The most common mistakes are not conceptual but bookkeeping: mismatched variable conventions, forgotten wrapper changes, wrong units, or altered accuracy settings that silently bias the answer.

Historically, many specialized codes have grown out of CAMB in exactly this way. A well-known example is EFTCAMB, which implements effective-field-theory descriptions of dark energy and modified gravity by extending the CAMB framework rather than replacing it [7]. Students do not need to learn that code here, but it is a good illustration of how a mature Boltzmann code becomes research infrastructure.

11. Common pitfalls

A short list of warnings saves a great deal of frustration:

- **Gauge confusion.** Internal variables in CAMB are not always the gauge variables written in a textbook, even when the physics is the same.
- k **versus** k/h**.** Matter-power outputs can use either convention; check the documentation and method options carefully [11].
- **Physical densities.** CAMB commonly wants `ombh2` and `omch2`, not just fractional densities Ω_b and Ω_c [12].
- **Forgetting to request matter outputs.** If you want $P(k, z)$, you usually need `WantTransfer = True` and an appropriate `set_matter_power()` call.
- **Changing physics without validation.** Any modification should first reproduce the standard result before being trusted in a new model.

12. Relation to parameter inference

Although this lecture is about CAMB rather than samplers, it is worth one final remark. In real analyses, CAMB is often the forward model inside a parameter-estimation pipeline such as CosmoMC or Cobaya [4, 13]. This is why it is so valuable for students to understand the code itself. Once the forward model is trusted, sampling and inference become conceptually much easier.

13. Summary

The slides were right to organize the lecture around three steps: initial conditions, evolution equations, and output statistics. In more polished language, CAMB works by:

(1) building the background and the primordial spectrum;
(2) imposing regular early-time initial conditions;
(3) evolving the coupled Einstein-Boltzmann system for each k mode;

(4) constructing source functions and transfer functions;
(5) projecting them into observable power spectra such as C_ℓ and $P(k)$.

The code looks complicated only because cosmology contains several interacting physical sectors. Once one learns the map between the files, the variable names, and the perturbation equations, CAMB becomes a readable and highly reusable piece of scientific software.

Homework

(1) Starting from the synchronous-gauge relation $\delta_c' = -h'/2$ and the CAMB definition $z = h'/(2k)$, show explicitly that the code equation `clxcdot = -k*z` is equivalent to the Ma-Bertschinger CDM continuity equation.
(2) Explain in words why the line-of-sight method is faster than evolving every multipole to the present time for every ℓ. In your answer, identify which part of the calculation depends mainly on geometry and which part depends on the cosmological model.
(3) In the tight-coupling era, why is it reasonable to set $\theta_b \simeq \theta_\gamma$ to leading order? What physical process enforces this relation?
(4) Write down a minimal CAMB workflow to compute both C_ℓ^{TT} and the linear matter power spectrum at $z = 0$. You do not need exact code syntax, but you should name the key objects or steps in the right order.

Suggested reading for students

For the code itself, start with the official CAMB documentation, the readme, and the theory page [11, 9, 14]. For perturbation theory conventions, read Ma and Bertschinger [1]. For the line-of-sight method, read Seljak and Zaldarriaga [2] and then the original CAMB paper [3]. For wider cosmological context, Hu and Dodelson's review remains very useful [5].

References

[1] C.-P. Ma and E. Bertschinger, "Cosmological Perturbation Theory in the Synchronous and Conformal Newtonian Gauges," *Astrophys. J.* **455**, 7-25 (1995), arXiv:astro-ph/9506072.
[2] U. Seljak and M. Zaldarriaga, "A Line-of-Sight Integration Approach to Cosmic Microwave Background Anisotropies," *Astrophys. J.* **469**, 437-444 (1996), arXiv:astro-ph/9603033.
[3] A. Lewis, A. Challinor, and A. Lasenby, "Efficient Computation of Cosmic Microwave Background Anisotropies in Closed Friedmann-Robertson-Walker Models," *Astrophys. J.* **538**, 473-476 (2000), arXiv:astro-ph/9911177.
[4] A. Lewis and S. Bridle, "Cosmological Parameters from CMB and Other Data: a Monte Carlo Approach," *Phys. Rev. D* **66**, 103511 (2002), arXiv:astro-ph/0205436.
[5] W. Hu and S. Dodelson, "Cosmic Microwave Background Anisotropies," *Ann. Rev. Astron. Astrophys.* **40**, 171-216 (2002), arXiv:astro-ph/0110414.
[6] A. Lewis and A. Challinor, "Weak Gravitational Lensing of the CMB," *Phys. Rept.* **429**, 1-65 (2006), arXiv:astro-ph/0601594.
[7] B. Hu, M. Raveri, N. Frusciante, and A. Silvestri, "Effective Field Theory of Cosmic Acceleration: an Implementation in CAMB," *Phys. Rev. D* **89**, 103530 (2014), arXiv:1312.5742 [astro-ph.CO].

[8] A. Lewis and A. Challinor, "CAMB GitHub repository," https://github.com/cmbant/CAMB, accessed May 2026.
[9] A. Lewis and A. Challinor, "CAMB Fortran readme," https://camb.info/readme.html, accessed May 2026.
[10] A. Lewis, "Fortran CAMB documentation," https://camb.info/doc/, accessed May 2026.
[11] A. Lewis, "CAMB Python documentation," https://camb.readthedocs.io/en/latest/, accessed May 2026.
[12] A. Lewis, "CAMB input parameter model documentation," https://camb.readthedocs.io/en/latest/model.html, accessed May 2026.
[13] A. Lewis, "CAMB modifying the code documentation," https://camb.readthedocs.io/en/latest/modifying_code.html, accessed May 2026.
[14] A. Lewis, "CAMB theory page," https://camb.info/theory.html, accessed May 2026.