Rescaling VT factors to match tabulated VT averages

R. O'Shaughnessy, D. Wysocki

The injection VT code produces $\langle VT \rangle$ for several reference populations, using MC. The population codes (which work with occasionally very narrow distributions) work with analytic approximations VT_{ana} to $VT(m_1, m_2, \chi_{1,z}, \chi_{2,z})$. We need a parameter-dependent correction factor $f(m_1, m_2, \chi_1, \chi_2)$ so $VT \simeq f VT_{ana}$, chosen so we reproduce the tabulated averages $\langle VT \rangle$.

I. ARGUMENT: LEAST SQUARES FOR MULTIPLICATIVE CORRECTION FACTOR

Notation For notational brevity, let x denote binary parameters $m_1, m_2, \chi_{1,z}, \chi_{2,z}$; let g be our analytic reference model for VT; let Λ denote model hyperparameters and $p(x|\Lambda)$ be distributions binary parmaeters give hyperparameters. Let $y_1 \ldots y_N$ denote the list of all possible values of $\langle VT \rangle$ computed by injections (including their Monte Carlo error $\sigma_1^2 \ldots \sigma_N^2$)

The injection codes compute $\langle VT \rangle \equiv \int dx p(x|\Lambda) VT_{true}(x)$, with some Monte Carlo error. Our analytic approximations, with a correction factor, compute

$$y = \langle VT \rangle = \int dx p(x|\Lambda) VT(x) f(x) \tag{1}$$

Least-squares approximation: We expand our proposed correction factor in basis functions $f(x) = \sum_\alpha F_\alpha(x) \lambda_\alpha$. We minimize the likelihood

$$\ln L = \text{const} - \sum_k (y_k - \sum_\alpha H_{k\alpha} \lambda_\alpha)^2 / 2\sigma_k^2 \tag{2}$$

where H is the precomputed matrix of weight "moments"

$$H_{k,\alpha} = \int dx p(x|\Lambda_k) VT(x) F_\alpha(x) \tag{3}$$

This standard least squares problem has a solution

$$\lambda = (H^T \gamma H)^{-1} H^T \gamma y \tag{4}$$

where γ is a diagonal inverse covariance matrix for the measurements (ie. diagonal elements are $1/\sigma_1^2, \ldots$)

Proposed correction factor: We only have of order 15^2 reference values, and we're performing a correction to VT (numerically tricky) rather than to $\ln VT$ (numerically more stable, but not permitting closed-form solutions). We propose low-order quadratic basis function in m_1, m_2

Potential problems: This doesn't guarantee f is positive-definite over the range of interest

Estimated correction factor: Using the VT values provided by the injection code, we estimated $\lambda_\alpha$and therefore the correction function $\sum_\alpha F_\alpha \lambda_\alpha$ for several choices of basis functions

II. CALIBRATION RESULTS

We measured the calibration factors using three choices of basis:

1. scalar: $\{1\}$
2. linear: $\{1, m_1, m_2\}$
3. quadratic: $\{1, m_1, m_2, m_1 m_2, m_1^2, m_2^2\}$

The coefficients measured for each of these are included in this DCC document, under the filenames:

1. `calibration_scalar.json`
2. `calibration_linear.json`
3. `calibration_quadratic.json`

These are in the JSON format (equivalent in this case to the notation for Python dict's and list's), for loading them in Python see the docs for `json.load` (`Python2.7`, `Python3.7`). We stored the full output of `numpy.linalg.lstsq` here, but to use the results all you'll need is the `"coeffs"` field, and possibly the `"basis"` field for convenience.

Note that for the calibration, we used semi-analytic VT's with a fiducial observing time of 1day, the `SimNoisePSDaLIGOEarlyHighSensitivityP1200087` PSD, and a detection threshold of $\rho > 8$ in one IFO. We used reweighted injection VT's with pyCBC used for the detection threshold, and the observing time equal to all of O1 and O2. If you use these numbers, be careful that you set $T = 1$day.

Example to load the JSON files in Python (should work in both Python 2 and 3). Also in this DCC document as `coeff_example.py`.

```
import json

## Uncomment the one you'd like to use.
#fname = calibration_scalar.json
#fname = calibration_linear.json
fname = calibration_quadratic.json

## Defining the basis functions in a lookup table, so we can use the
## "basis" field in the JSON files to figure out which one to use,
## rather than have to hard-code anything.
basis_scalar = [
    lambda m1, m2, a1z, a2z: 1.0,
]
basis_linear = basis_scalar + [
    lambda m1, m2, a1z, a2z: m1,
    lambda m1, m2, a1z, a2z: m2,
]
basis_quadratic = basis_linear + [
    lambda m1, m2, a1z, a2z: m1*m2,
    lambda m1, m2, a1z, a2z: m1**2,
    lambda m1, m2, a1z, a2z: m2**2,
]
bases = {
    "scalar" : basis_scalar,
    "linear" : basis_linear,
    "quadratic" : basis_quadratic,
}

with open(fname, "r") as calibration_file:
    calibration_info = json.load(calibration_file)
    coeffs = calibration_info["coeffs"]
    basis_fns = bases[calibration_info["basis"]]

def f(m1, m2, a1z, a2z):
    """
    This is the correction factor function.
    Now anytime you evaluate VT(m1, m2, a1z, a2z), be sure to multiply
    by f(m1, m2, a1z, a2z), so
    vt_corrected = VT(m1, m2, a1z, a2z) * f(m1, m2, a1z, a2z)
    """
    return sum(
        c * g(m1, m2, a1z, a2z)
        for c, g in zip(coeffs, basis_fns)
    )
```