g2tools Documentation

Release 1.3.1

G.P. Lepage

Feb 15, 2018

CONTENTS

Contents:

CHAPTER

ONE

MUON G-2 FROM LATTICE QCD USING G2TOOLS

Module *g2tools* contains a small number of tools useful for analyzing contributions to the muon's magnetic moment from (lattice) QCD vacuum polarization. These tools were developed by G.P. Lepage to implement the analysis presented in Chakraborty *et al*, Phys.Rev. D89 (2014) no.11, 114501 (arXiv:1403.1778) and subsequent papers by the same authors.

A typical application, illustrating the most important tools, is provided by the following code:

```
import g2tools as g2
import gvar as gv

def main():
    # data
    Z = gv.gvar('0.9938(17)')           # current Z factor
    Q = 1. / 3.                         # charge of quark (units of proton charge)
    ainv = gv.gvar('1.6280(86)')        # inverse lattice spacing (in GeV)

    G = gv.gvar([                       # G(t) for t=0..63 (in lattice units)
        '0.0870904(11)', '0.0435138(14)', '0.00509859(48)', '0.00305614(43)',
        '0.00069516(19)', '0.00045466(15)', '0.000166972(80)', '0.000102219(58)',
        '0.000045284(34)', '0.000026213(22)', '0.000012630(14)', '7.0635(91)e-06',
        '3.5569(57)e-06', '1.9469(37)e-06', '1.0027(24)e-06', '5.421(16)e-07',
        '2.834(10)e-07', '1.5174(67)e-07', '7.943(43)e-08', '4.253(28)e-08',
        '2.221(19)e-08', '1.183(12)e-08', '6.132(81)e-09', '3.292(51)e-09',
        '1.727(34)e-09', '9.19(22)e-10', '4.81(14)e-10', '2.643(96)e-10',
        '1.385(64)e-10', '7.61(44)e-11', '3.92(31)e-11', '2.67(24)e-11',
        '2.07(21)e-11', '2.90(23)e-11', '4.12(31)e-11', '8.20(42)e-11',
        '1.380(65)e-10', '2.788(98)e-10', '5.01(15)e-10', '9.72(23)e-10',
        '1.782(34)e-09', '3.406(53)e-09', '6.333(78)e-09', '1.212(12)e-08',
        '2.249(18)e-08', '4.283(28)e-08', '8.016(44)e-08', '1.5263(67)e-07',
        '2.843(10)e-07', '5.420(16)e-07', '1.0062(25)e-06', '1.9453(39)e-06',
        '3.5611(58)e-06', '7.0675(93)e-06', '0.000012647(14)', '0.000026240(22)',
        '0.000045282(32)', '0.000102285(56)', '0.000166993(79)', '0.00045479(15)',
        '0.00069503(19)', '0.00305647(42)', '0.00509870(47)', '0.0435158(14)'
        ])
    # N.B.: In general would construct G so that correlations from one t
    #   to the next are included. Don't bother here since this is meant
    #   just to illustrate g2tools.

    # compute moments, converting to physical units from lattice units
    mom = g2.moments(G, ainv=ainv, Z=Z, periodic=True, nlist=[4, 6, 8, 10])
    print('Taylor coefficients:', g2.mom2taylor(mom))
    print()

    # construct subtracted vac pol function using [2,2] Padé
    vpol = g2.vacpol(mom, order=(2,2))
```

```
    # integrate vpol to get a_mu and print result
    amu = g2.a_mu(vpol, Q=Q)
    print('a_mu contribution =', amu)
    print()

    # error budget for a_mu
    print(gv.fmt_errorbudget(
        outputs=dict(a_mu=amu, mom4=mom[4]),
        inputs=dict(G=G, Z=Z, ainv=ainv),
        ))

if __name__ == '__main__':
    main()
```

In this code, we first read the simulation data for the *jj* correlator into array `G`, where `G[i]` is the correlator for (Euclidean) time `i/ainv` where `i=0,1..63`. We then use *g2tools.moments()* to calculate temporal moments of the correlator, while also converting from lattice units to physical units (using the inverse lattice spacing `ainv`) and renormalizing the current (`Z`).

`vpol(q2)` is the vacuum polarization function at Euclidean q^2 equal to `q2`. Object `vpol` has type *g2tools.vacpol*. It constructs a [2,2] Padé approximant from the moments, and uses that approximant to approximate the exact function. The approximants converge to the exact result as the order increases provided the momentum is space-like (`q2` non-negative). Using a [1,1] Padé instead of [2,2] gives almost identical results here, so the approximants have converged for the present application.

We calculate the contribution from vacuum polarization `vpol` to the muon's anomalous magnetic moment a_μ using `g2tools.a_mu()`. We also use `gvar.fmt_errorbudget()` to produce an error budget for it and the 4th moment.

Running this code gives the following output:

```
Taylor coefficients: [0.06629(74) -0.0527(11) 0.0472(15) -0.0435(18)]

a_mu contribution = 5.412(57)e-09

Partial % Errors:
                a_mu      mom4
-------------------------------
     ainv:      1.00      1.06
        Z:      0.34      0.34
        G:      0.01      0.01
-------------------------------
    total:      1.06      1.11
```

The contribution to the muon's anomalous magnetic moment is $54.12(57)\text{x}10^{-10}$. The error budget shows that the final uncertainty is dominated by the uncertainty in the inverse lattice spacing `ainv`; statistical errors from `G` are completely negligible in this example.

An alternative to using moments is to Fourier transform the correlator to obtain `vpol(q2)` directly. Moments are particularly useful for analyzing finite-volume and other systematic errors, but the Fourier method is simpler to code:

```
# compute a_mu from the Fourier transform of G(t)
vpol = g2.fourier_vacpol(G, ainv=ainv, Z=Z, periodic=True)
amu = g2.a_mu(vpol, Q=Q)
print('a_mu contribution =', amu)
print()
```

```
# error budget for a_mu
print(gv.fmt_errorbudget(
    outputs=dict(a_mu=amu),
    inputs=dict(G=G, Z=Z, ainv=ainv),
    ))
```

This code gives identical results to that above.

g2tools is designed to work with module `gvar` which we use here to represent the statistical and systematic uncertainties in the correlator values, inverse lattice spacing, and `Z` factor. Each of these quantities is an object of type `gvar.GVar`, which represents a Gaussian random variable. `gvar.GVars` describe not only means and standard deviations, but also statistical correlations between different objects. These correlations are propagated through arbitrary arithmetic statements. Adding the following code to the end of `main()`, for example,

```
print(gv.evalcorr([mom[4], mom[6], mom[8], mom[10]]))
```

prints out the correlation matrix for the moments, showing that they are highly correlated (as expected):

```
[[ 1.          0.98833867  0.9787737   0.97262094]
 [ 0.98833867  1.          0.99853653  0.99646438]
 [ 0.9787737   0.99853653  1.          0.99949934]
 [ 0.97262094  0.99646438  0.99949934  1.        ]]
```

The moments are also highly correlated with the final results `a_mu`: for example, adding the following to the end of `main()`

```
print(gv.evalcorr([a_mu, mom[4]]))
```

gives:

```
[[ 1.          0.96864247]
 [ 0.96864247  1.        ]]
```

This kind of correlation information is used by `gvar.fmt_errorbudget(...)` to create the error budget. See `gvar`'s documentation for more information.

CHAPTER

TWO

G2TOOLS MODULE

2.1 Moments

The main tools for creating and manipulating moments are:

g2tools.**moments**(*G, Z=1.0, ainv=1.0, periodic=True, tmin=None, nlist=[4, 6, 8, 10, 12, 14, 16, 18, 20]*)
Compute t**n moments of correlator G.

Compute `sum_t t**n G(t)` for `n` in `nlist`, where both positive and negative `t` are included.

Parameters

- **G** – Array of correlator values `G[t]` for `t=0,1...` (in lattice units).
- **Z** – Renormalization factor for current (moments multiplied by `Z**2`). Defaul is 1.
- **ainv** – Inverse lattice spacing used to convert moments to physical units (n-th moment multiplied by `1/ainv**(n-2)`). Default is 1.
- **periodic** – `periodic=True` implies `G[-t] = G[t]` (default); `periodic=False` implies no periodicity in array `G[t]` (and results doubled to account for negative `t`).
- **tmin** – minimum `t` value included in moments; ignored if `None` (default).
- **nlist** – List of moments to calculate. Default is `nlist=[4,6,8...20]`.

Returns Dictionary `Gmom` where `Gmom[n]` is the `n-th` moment.

g2tools.**mom2taylor**(*mom*)
Convert moments in dictionary `mom` into Taylor series coefficients.

g2tools.**taylor2mom**(*tayl*)
Convert Taylor coefficients in array `tayl` to moments.

2.2 Subtracted Vacuum Polarization

A subtracted vacuum polarization function (`Pi-hat`) is represented by the following classes:

class g2tools.**vacpol**(*g*, *order=None*, *scale=None*, *rtol=None*, *qth=0*, *warn=True*, *exceptions=True*)
Subtracted vac. pol'n (`Pi-hat(q2)`) from correlator moments `Gmon[n]`.

The current-current correlator is `q2 * Pi(q2)`, where `Pi-hat(q2) = Pi(q2) - Pi(0)` is the subtracted (i.e., renormalized) vacuum polariztion function.

The vacuum polarization function is a Pade approximant to the Taylor series corresponding to the moments `g[n]`. The code estimates the precision of the moments and sets the tolerance for the Pade determination

accordingly. The order `(m,n)` of the Pade can be specified, but might be reduced by the code if the data are noisy.

vacpol objects are used primarily as functions (of q^2) but also have several attributes. Attribute `pseries` is a dictionary containing various powerseries (see `gvar.powerseries`) describing the function: the vacuum polarization function is `q2` times a Pade approximant with a numerator given by `pseries['num']` and a denominator given by `pseries['den']`. The Taylor series for this function is given by `q2` times `pseries['taylor']`.

vacpol objects also have a method *vacpol.badpoles()* that tests the poles in the denomator of the Pade. `badpoles(qth)` returns `False` if any of the poles is complex or if any are located above `-(qth ** 2)`. `qth` should be set equal to the threshold energy for the correlator. If it is unset, `qth=0` is used. Lists of the poles and their residues (for `Pi-hat(q2)`) are available in attributes `pole` and `residue`, respectively.

vacpol has several static methods for creating specialized examples of vacuum polarizations (e.g., for testing):

- `vacpol.fermion(m)` – 1-loop fermion (mass `m`) contribution;
- `vacpol.scalar(m)` – 1-loop scalar (mass `m`) contribution;
- **`vacpol.vector(m, f)` – tree-level contribution from vector** with mass `m` and decay constant `f`.

  Parameters

  - **`g`** – Dictionary containing moments where `g[n] = sum_t t**n G(t)`, or array containing Taylor coefficients where `Pi-hat(q2) = q2 * sum_j q2**j * g[j]`.
  - **`order`** – Tuple `(m,n)` specifying the order of the Pade approximant used to approximate `Pi-hat(q2)` (the function is approximated by `q2` times an `(m-1,n)` approximant). The order may be reduced (automatically) if the data are too noisy. If the order is not specified, it is set automatically according to the number of entries in `G`.
  - **`scale`** – Scale factor used to rescale `q2` so that the Taylor coefficients are more uniform in size. This is normally set automatically (from the first two moments), but the automatic value is overridden if `scale` is set.
  - **`rtol`** – Relative tolerance assumed when determining the Pade approximant. This is normally set automatically (from the standard deviations of the moments), but the automatic value is overridden if `rtol` is specified.
  - **`qth`** – Threshold for particle production: poles above `-qth**2` are bad. Default is `qth=0`.
  - **`warnings`** – `warnings=True` (default) causes a warning to be issued when the order has been reduced automatically. `warnings=False` suppresses the warnings.
  - **`exceptions`** – If `True` (default), an exception is raised if there are bad poles in the `vacpol`. If `False`, exceptions are suppressed.

Methods include:

`taylor`(*n=None*)
Return Taylor coefficients for `PI-hat(q2)/q2`.

Parameters **`n`** – Maximum number of coefficients returned. Returns all coefficents if `None` (default)/

`badpoles`(*qth=None*)
True if any pole is complex or above threshold.

Parameters **`qth`** – Threshold for particle production: poles above `-qth**2` are bad. (Default is `qth=0`.)

FT(*t*, *ainv=1.0*)
Fourier transform of `q2 * PI-hat(q2)`.

The Pade approximant can be decomposed into a sum of poles (partial fractions), which give a sum of decaying exponentials when Fourier transformed back to t-space. The amplitudes and energies of these exponentials (for the transform of `q2 * Pi-hat(q2)''') are stored in :class:`g2tools.vacpol` attributes ``E` and `ampl`, respectively.

The decomposition into a sum of poles leaves a residual polynomial in `q2` (zeroth-order for `(n,n)` Pades). This is ignored in the Fourier transform since it typically affects the transform only for very small `t`. These terms have a negligible effect (suppressed by `a**2j` on the Taylor coefficients `Pi[j]` of `Pi-hat(q2)` (for j>=1).

Optional parameter `ainv` can be used to convert the Fourier transform to lattice units (by multiplying it by `1/ainv**3`) for comparison with simulation data. The times `t` are then assumed to be in lattice units.

Parameters

- **t** (*number, array*) – Time in physical units unless `ainv` is specified, in which case lattice units are assumed.
- **ainv** – Inverse lattice spacing. The Fourier transform is in lattice units if `ainv` is specified (assuming the original Taylor coefficients are in physical units).

static scalar(*m*, *n=10*, *use_pade=False*)
1-loop subt. vac. pol'n from a scalar with mass m (and charge=1).

static fermion(*m*, *n=19*, *use_pade=False*)
1-loop subt. vac. pol'n from a fermion with mass m (and charge=1).

static vector(*m*, *f=1.0*, *n=10*, *use_pade=False*)
Vac. pol'n due to a vector with mass `m` and decay const. `f`.

The decay constant is defined such that the vacuum polarization function is `Pi-hat = q2 * f**2/2/m**2 / (q2 + m**2)`. This corresponds in `t` space to ‘‘m * f**2 * exp(-m * t) / 4’‘.

class g2tools.**fourier_vacpol**(*G*, *Z=1.0*, *ainv=1.0*, *periodic=True*)
Subtracted vac. pol'n (`Pi-hat(q2)`) from correlator `G(t)`.

The correlator is Fourier transformed to produce a function `Pi_hat` of (Euclidean) q^2 suitable for use in `g2tools.a_mu()`.

See Bernecker & Meyer, EPJA47 (2011) 148 , arXiv:1107.4388 for details on the Fouier transformation.

Parameters

- **G** (*array*) – Current-current correlator in an array whose elements are `[G(0), G(a),G(2*a),...,G(-2*a),G(-a)]` if `periodic=True` or `[G(0),G(a),...,G(T*a-1)]` otherwise. `G` is assumed to be in lattice units.
- **Z** – Renormalization factor for current (correlator multiplied by `Z**2`). Defaul is 1.
- **ainv** – Inverse lattice spacing used to convert Fourier transform to physical units. Default is 1.
- **periodic** – `periodic=True` implies `G[-t] = G[t]` (default); `periodic=False` implies `G[t]` is not periodic and is specified for only non-negative `t` values (results are doubled to account for negative `t`).

2.3 Padé Approximants

The following two functions are used for calculating Padé approximants from the Taylor coefficients of an arbitrary function. The first (*g2tools.pade_svd()*) implements an algorithm that uses *svd* cuts to address instabilities caused by uncertainties in the Taylor coefficients. The second function (*g2tools.pade_gvar()*) is built on the first but allows Taylor coefficients to have uncertainties (`gvar.GVars`). The statistical uncertainties and correlations between different coefficients are propagated through the analysis.

`g2tools.`**`pade_svd`**(*f*, *m*, *n*, *rtol=1e-14*)

`[m,n]` Pade approximant to `sum_i f[i] x**i`.

The `[m,n]` Pade approximant to a series given by `sum_i f[i] * x**i` is the ratio of polynomials of order `m` (numerator) and `n` (denominator) whose Taylor expansion agrees with that of the original series up to order `m+n`.

This code is adapted from P. Gonnet, S. Guttel, L. N. Trefethen, SIAM Review Vol 55, No. 1, 101 (2013). It uses an *svd* algorithm to deal with imprecision in the input data, here specified by the relative tolerance `rtol` for the input coefficients `f[i]`. It automatically reduces the order of the approximant if the extraction of Pade coefficients is too unstable given tolerance `rtol`.

Parameters

- **`f`** – Array `f[i]` of power series coefficients for `i=0...n+m`.
- **`m`** – Maximum order of polynomial in numerator of Pade approximant (`m>=0`).
- **`n`** – Maximum order of polynomial in denominator of Pade approximant (`m>=0`).
- **`rtol`** – Relative accuracy of input coefficients. (Default is 1e-14.)

Returns Tuple of power series coefficients `(p, q)` such that `sum_i p[i] x**i` is the numerator of the approximant, and `sum_i q[i] x**i` is the denominator. `q[0]` is normalized to 1.

`g2tools.`**`pade_gvar`**(*f*, *m*, *n*, *rtol='gavg'*)

`[m,n]` Pade approximant to `sum_i f[i] x**i` for `GVars`.

The `[m,n]` Pade approximant to a series given by `sum_i f[i] * x**i` is the ratio of polynomials of order `m` (numerator) and `n` (denominator) whose Taylor expansion agrees with that of the original series up to order `m+n`.

This code uses an SVD algorithm (see *pade_svd()*) to deal with imprecision in the input data. It automatically reduces the order of the approximant if the extraction of Pade coefficients is too unstable given noise in the input data.

Parameters

- **`f`** – Array `f[i]` of power series coefficients for `i=0...n+m`.
- **`m`** – Maximum order of polynomial in numerator of Pade approximant (`m>=0`).
- **`n`** – Maximum order of polynomial in denominator of Pade approximant (`m>=0`).
- **`rtol`** (*`float or str`*) – If `rtol` is a string, it determines how the relative tolerance is determined from the relative uncertainties in the `f[i]`. Set `rtol` equal to: `'gavg'` for the geometric mean (default); `'avg'` for the average; `'min'` for the minimum; or `'max'` for the maximum. Otherwise a number can be specified, in which case the uncertainties in `f[i]` are ignored.

Returns Tuple of power series coefficients `(p, q)` such that `sum_i p[i] x**i` is the numerator of the approximant, and `sum_i q[i] x**i` is the denominator. `q[0]` is normalized to 1.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX