# Chapter 7: the static quark potential and spectrum of pure QCD

In this chapter we study for the first time the physics of pure QCD.

## Temporal Gauge

Let us identify $\mu = 3$ with the time direction and consider a lattice with infinite time dimension. A gauge transformation allows us to select a $V(x) \in$ SU(3) for every Euclidean lattice site $x$. By picking

$$V(\vec{x}, t=0) = 1\,, \tag{1}$$
$$V(\vec{x}, t=1) = U_3(\vec{x}, t=0)\,, \tag{2}$$
$$V(\vec{x}, t=2) = U_3(\vec{x}, t=0)U_3(\vec{x}, t=1)\,, \tag{3}$$
$$\ldots \tag{4}$$
$$V(\vec{x}, t=-1) = U_3(\vec{x}, t=-1)^\dagger\,, \tag{5}$$
$$V(\vec{x}, t=-2) = U_3(\vec{x}, t=-1)^\dagger U_3(\vec{x}, t=-2)^\dagger\,, \tag{6}$$
$$\ldots \tag{7}$$

we can transform all temporal links to

$$U_3(x,t) = 1 \tag{8}$$

which is the **temporal gauge** condition. This is a special version of a **tree gauge**, where one selects the $V(x)$ such that they set all links on a tree starting from a given origin site to the identity. The action and integration measure are invariant under this transformation, however, it is particularly useful in interpreting the system in a Hamiltonian sense in which the time direction is special as we will see below.

## Static Quarks

We remind ourselves that the Lagrangian of a free quark with mass $m$ in Minkowski space is given by

$$\mathcal{L} = \overline{\psi}(i\gamma^\mu\partial_\mu - m)\psi\,. \tag{9}$$

In a theory for sufficiently large mass $m$, the temporal derivatives $\partial_0$ and therefore also $\partial_0$ must scale as

$$\partial_0\psi = i(m + \frac{p^2}{2m} + O(1/m^2))\psi \tag{10}$$

with spatial momentum $p$. One finds that an appropriate action for the infinitely heavy quark ($m \to \infty$) is

$$\mathcal{L} = \overline{h}i\gamma_0\partial_0 h + O(1/m)\,, \tag{11}$$

where the field $h$ has to be appropriately defined. Such a **static** quark no longer has a spatial derivative term, which means that it can no longer propagate in space. One can show that on a lattice, the propagator of such a static quark from $x$ to $x + \hat{3}n$ is given by the Wilson line

$$U_3(x)U_3(x+\hat{3})\cdots U_3(x+\hat{3}(n-1))\,. \tag{12}$$

## Static Quark Potential

Since our gauge action is invariant under axis interchange, we can also consider static quarks that propagate in a given spatial direction. If we now considered a Wilson loop $W(r,t)$ of extend $r$ spatial sites and $t$ temporal sites in temporal gauge, we could identify $W(r,t)$ as two static quarks propagating the distance $r$ separated by $t$ time steps. Another way to think about this is that we create a state of a static quark propagating distance $r$ and project to it by inserting the transition matrix $e^{-Ht}$ in between. The energy of such a state

$$V(r) = \lim_{t\to\infty} \log \frac{W(r,t)}{W(r,t+1)} \tag{13}$$

is also called the static quark potential.

From perturbative QCD, we know that the effective coupling constant $g$ at short distances vanishes in QCD, i.e., at short distances the QCD action's field strength tensor

$$F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu + ig[A_\mu, A_\nu] \tag{14}$$

goes over to 8 separate contributions (number of SU(3) generators) that look like the electromagnetic field strength tensor

$$F^{\text{em}}_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu\,. \tag{15}$$

We should therefore expect QCD to behave similarly to electrodynamics at very short distances, i.e., we would expect a Coulomb potential $V(r) \propto 1/r$ at short distances.

In the next chapter, we will systematically study the strong coupling expansion of QCD and will find that in this limit, we will expect a linear rise in the potential which we can identify with the long-distance behavior $V(r) \propto r$.

In total, a reasonable ansatz for $V(r)$ is

$$V(r) = A + \frac{B}{r} + \sigma r\,. \tag{16}$$

We will confirm this behavior below by explicit numerical study. The long distance behavior also tells us that the force

$$F(r) = V'(r) \tag{17}$$

between to static quarks, will be constant for sufficiently large $r$. This constant force is captured by the **string tension** $\sigma$ and signals quark **confinement**, i.e., no matter how far we separate the static quarks, they will always have a restoring force that binds them together. The unusual sign definition used here allows for a more convenient sign definition in the next subsection.

By introducing dynamical quarks to the theory, the energy stored in this potential may be sufficient to create a quark-antiquark pair that can then break the string, more on this at the appropriate chapter.

## Sommer scale

By also studying the spectra of bottom and charm quark states and comparing them to experimentally measured results one can determine that the force between static quarks at **Sommer scale**

$$r_0 \approx 0.5\text{fm} \tag{18}$$

satisfies

$$r_0^2 F(r_0) = 1.65\,. \tag{19}$$

This was one of the first means to determine the physical lattice spacing given a set of ensemble parameters. We will illustrate its use below. We will also discuss more modern techniques in later chapters.

## Glueball spectrum

The spectrum of pure QCD contains bound states of gluons or **glueballs**. This is possible due to the self-interaction of SU(3) gauge theory, see comparison of the field-strength tensor with the U(1) field strength tensor above.

The spectrum of such states can be categorized using good quantum numbers obtained by studying the irreducible representations of the spacetime symmetry group of the lattice and the continuum. We shall discuss this also in later chapters more systematically. For now, we learn that the trivial scalar representation that is also invariant under parity and flipping the charge $g \to -g$, called $A_1^{++}$ representation, contains the lightest glueball state.

Any operator transforming in this irreducible representation will excited such a particle. The simplest such operator is the sum over all spatial plaquettes $P_{ij}$ with $i, j < 3$. Note that since the vacuum expectation value of the plaquette does not vanish, we need to subtract the expectation value of the plaquette when constructing a correlator to measure its mass. This is identical to the case of the $\phi^2$ operator in the real scalar theory studied in chapter 4.

## Reduction of excited state contributions by gauge smearing

In the operators constructed below, we replace the original gauge fields by gauge fields that are smeared out over a finite spatial distance, while still satisfying the gauge transformation property

$$U_\mu(x) \to V(x) U_\mu(x) V(x+\hat{\mu})^\dagger\,. \tag{20}$$

This merely creates a different operator in the same irreducible representation but has the effect of suppressing overlap with the more short-distance excited states.

Note: in this chapter, we will use the relation between units $0.1973\ \text{GeV fm} = 1$.

In [2]:

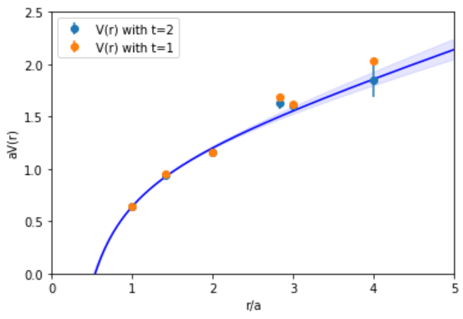
```python
import gpt as g;
import numpy as np;
from matplotlib import pyplot as plt;

def generate_ensemble(Ls, Lt, beta, Nthermalize, Nmax, Nskip):
    L = [Ls, Ls, Ls, Lt]

    grid = g.grid(L, g.double)
    grid_eo = g.grid(L, g.double, g.redblack)

    rng = g.random("test", "vectorized_ranlux24_24_64")
    U = g.qcd.gauge.unit(grid)
    Nd = len(U)

    mask_rb = g.complex(grid_eo)
    mask_rb[:] = 1

    # full mask
    mask = g.complex(grid)

    def staple(U, mu):
        st = g.lattice(U[0])
        st[:] = 0
        Nd = len(U)
        for nu in range(Nd):
            if mu != nu:
                st += g.qcd.gauge.staple(U, mu, nu) / U[0].otype.Nc
        return st

    g.default.set_verbose("su2_heat_bath", False)
    markov = g.algorithms.markov.su2_heat_bath(rng)
    plaquette_heatbath = []
    for it in range(Nmax):
        plaq = g.qcd.gauge.plaquette(U)
        plaquette_heatbath.append(plaq)
        if it % (Nmax//20) == 0:
            g.message(f"SU(2)-subgroup heatbath {it} has P = {plaq}")
        for cb in [g.even, g.odd]:
            mask[:] = 0
            mask_rb.checkerboard(cb)
            g.set_checkerboard(mask, mask_rb)

            for mu in range(Nd):
                st = g.eval(beta * staple(U, mu))
                markov(U[mu], st, mask)
        if it > Nthermalize and it % Nskip == 0:
            g.default.push_verbose("io", False)
            g.save(f"{Ls}c{Lt}_{beta}/su3.{it}",U)
            g.default.pop_verbose()


    fig, ax = plt.subplots()
    ax.plot(range(len(plaquette_heatbath)), plaquette_heatbath, marker='+', ls='', c='red', label="Heatbath")
    plt.legend()
    plt.show()

#generate_ensemble(8, 32, 5.5, 100, 10000, 5)
#generate_ensemble(8, 8, 5.9, 50, 300, 5)
#generate_ensemble(8, 8, 5.5, 100, 300, 5)
#for beta in [5.6,5.7,5.8]:
#    generate_ensemble(8, 8, beta, 100, 300, 5)
```

In [3]:

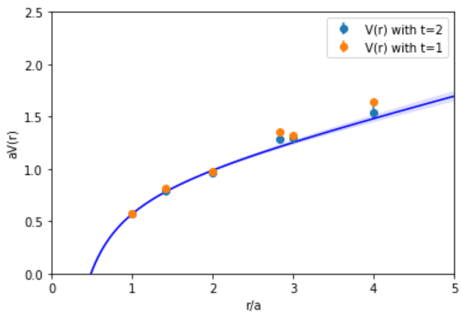
```
from scipy.optimize import curve_fit

g.default.set_verbose("io", False)

def jackknife_covariance_estimator(f, x):
    N = len(x)
    X = sum(x) / N
    mean = f(X)
    def outer_sqr(a):
        return np.outer(a,a)
    return sum([outer_sqr(f((N*X - x[j])/(N-1)) - mean) for j in range(N)])*(N-1)/N

def V(t0, configurations):
    U = g.load(configurations[0])

    Wt = [{},{}]

    for t in [t0,t0+1]:
        for r in range(1,5):
            Wt[t-t0][r] = g.qcd.gauge.transport(U,[g.qcd.gauge.path().f(3,t).f(nu,r).b(3,t).b(nu,r)
                                  for nu in range(3)])
        for r in [1,2]:
            Wt[t-t0][np.sqrt(2.)*r] = g.qcd.gauge.transport(U,
                                            [g.qcd.gauge.path().f(3,t).f(nu,r).f(rho,r).b(3,t).b(rho,r).b(nu,r)
                                             for nu in range(3) for rho in range(3) if nu != rho])

    Wlc = []
    for conf in configurations:
        U = g.load(conf)

        for i in range(4):
            U = g.qcd.gauge.smear.stout(U, rho=1.0/12.0,orthogonal_dimension=3)

        print(conf, end="\r")
        Wlc.append([[sum([g.sum(g.trace(l)).real for l in wti[r](U)]) for wti in Wt] for r in Wt[0]])

    Wlc = np.array(Wlc)
    Wl_mean = np.mean(Wlc,axis=0)
    V_mean = [np.log(Wl_mean[i,0]/Wl_mean[i,1]) for i in range(len(Wl_mean))]
    rs = [r for r in Wt[0]]
    V_cov = jackknife_covariance_estimator(lambda x: np.array([np.log(x[i,0]/x[i,1]) for i in range(len(Wl_mean))]),
                                          Wlc)
    V_err = [V_cov[i,i]**0.5 for i in range(len(V_mean))]
    print()
    return rs, V_mean, V_err, V_cov

def lattice_spacing(configurations):
    def model(r,*p):
        return p[0] + p[1]/r + p[2]*r

    def model_gradient(r,*p):
        return np.array([r**0,r**-1,r]).T

    rs1, Vm1, Ve1, Vc1 = V(1, configurations)
    rs2, Vm2, Ve2, Vc2 = V(2, configurations)

    par_mean, par_cov = curve_fit(model, rs2, Vm2, p0=[0.5,0.5,0.5], sigma=Vc2, absolute_sigma=True, jac=model_gradient)
    a_in_fm = ((par_mean[1] + 1.65)/par_mean[2])**-0.5 * 0.5

    a_in_fm_dpar1 = -0.5 * ((par_mean[1] + 1.65)/par_mean[2])**-1.5 * 0.5 * 1.0 / par_mean[2]
    a_in_fm_dpar2 = 0.5 * ((par_mean[1] + 1.65)/par_mean[2])**-1.5 * 0.5 * 1.0 * (par_mean[1] + 1.65) / par_mean[2]**2
    a_in_fm_dpar = np.array([0,a_in_fm_dpar1,a_in_fm_dpar2])

    a_in_fm_err = np.dot(a_in_fm_dpar,np.dot(par_cov,a_in_fm_dpar))**0.5
    print(a_in_fm, a_in_fm_err)

    def err_f(r,p,cov_p):
        return np.dot(model_gradient(r,*p), np.dot(cov_p, model_gradient(r,*p)))**0.5

    rrange = np.arange(0.1, 5.0, 0.01)
    fy = np.array([model(r,*par_mean) for r in rrange])
    fyerr = np.array([err_f(r,par_mean,par_cov) for r in rrange])

    fig, ax = plt.subplots()

    plt.ylim(0,2.5)
    plt.xlim(0,5)
    plt.xlabel("r/a")
    plt.ylabel("aV(r)")
    ax.fill_between(rrange,fy-fyerr,fy+fyerr,alpha=0.1,color="blue")
    ax.plot(rrange,fy,c="blue")

    ax.errorbar(rs2, Vm2, Ve2, fmt="o", label="V(r) with t=2")
    ax.errorbar(rs1, Vm1, Ve1, fmt="o", label="V(r) with t=1")

    plt.legend()
    plt.show()

    return (a_in_fm, a_in_fm_err, rs2, Vm2, Ve2, par_mean, par_cov)
```

In [4]:
```
betas = [5.5,5.6,5.7,5.8,5.9]
a_mean_err = []
for beta in betas:
    a_mean_err.append(lattice_spacing([f"8c8_{beta}/su3.{it}" for it in range(105,300,5)]))
```

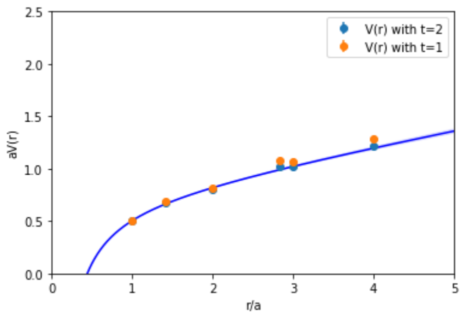
```
8c8_5.5/su3.295
8c8_5.5/su3.295
0.24683174310693679 0.01064766513162157
```



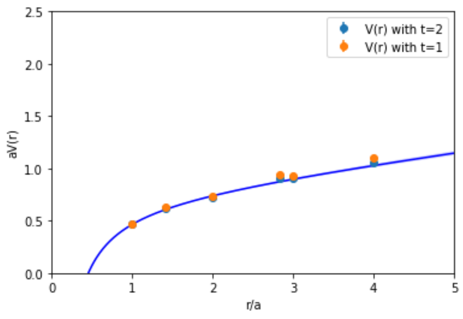
```
8c8_5.6/su3.295
8c8_5.6/su3.295
0.19966713489661383 0.0060758674234884765
```



```
8c8_5.7/su3.295
8c8_5.7/su3.295
0.16705357835336412 0.003082202050203599
```

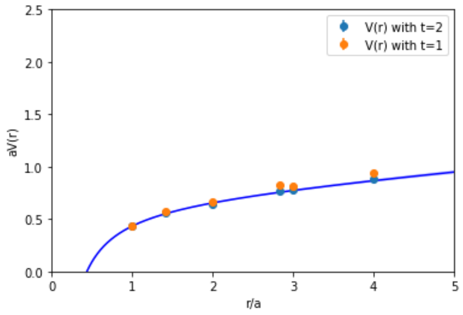


```
8c8_5.8/su3.295
8c8_5.8/su3.295
0.13957822686420235 0.0031690294006383735
```

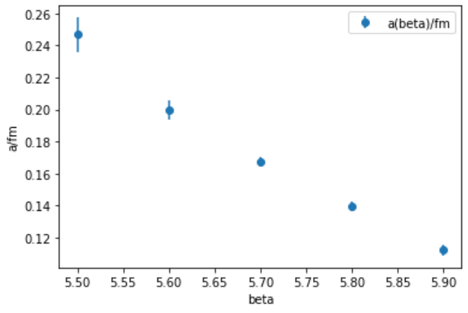


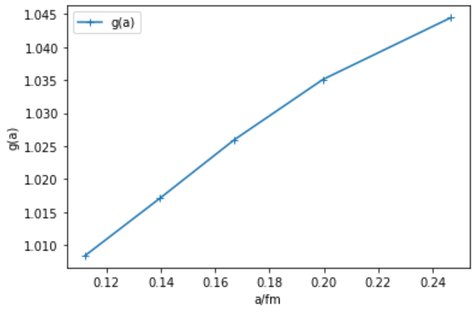
```
8c8_5.9/su3.295
```

```
8c8_5.9/su3.295
0.11210492116697537 0.003412302400457159
```

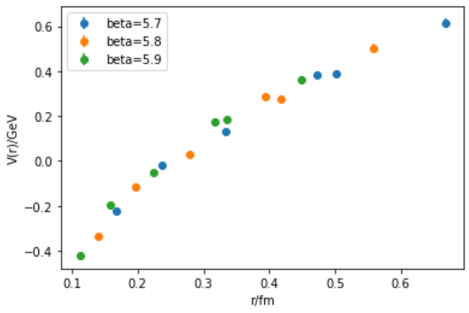


In [11]:
```
fig, ax = plt.subplots()

plt.xlabel("beta")
plt.ylabel("a/fm")

ax.errorbar(betas, [a[0] for a in a_mean_err], [a[1] for a in a_mean_err], fmt="o", label="a(beta)/fm")

plt.legend()
plt.show()

fig, ax = plt.subplots()

plt.xlabel("a/fm")
plt.ylabel("g(a)")

ax.plot([a[0] for a in a_mean_err], [(6/b)**0.5 for b in betas], marker="+", label="g(a)")

plt.legend()
plt.show()
```

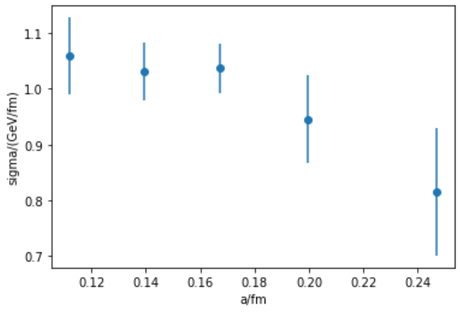




In [13]:
```
fig, ax = plt.subplots()
plt.xlabel("r/fm")
plt.ylabel("V(r)/GeV")

# merge all

for i in range(2,len(betas)):
    r_in_fm = []
    V_in_GeV = []
    V_in_GeV_err = []
    for j in range(len(a_mean_err[i][2])):
        r_in_fm.append(a_mean_err[i][0]*a_mean_err[i][2][j])
        V_in_GeV.append((a_mean_err[i][3][j]-a_mean_err[i][5][0])/a_mean_err[i][0]*0.1973)
        V_in_GeV_err.append(a_mean_err[i][4][j]/a_mean_err[i][0]*0.1973)
    ax.errorbar(r_in_fm, V_in_GeV, V_in_GeV_err, fmt="o", label=f"beta={betas[i]}")

plt.legend()
plt.show()

# Show also coarser
```



In [61]:
```
fig, ax = plt.subplots()
plt.xlabel("a/fm")
plt.ylabel("sigma/(GeV/fm)")

# merge all
a_in_fm = []
sigma_in_GeV = []
sigma_in_GeV_err = []
for i in range(len(betas)):
    a_in_fm.append(a_mean_err[i][0])
    sigma_in_GeV.append(a_mean_err[i][5][2]/a_mean_err[i][0]**2.*0.1973)
    sigma_in_GeV_err.append(a_mean_err[i][6][2,2]**0.5/a_mean_err[i][0]**2.*0.1973)

    #sigma_in_GeV.append(a_mean_err[i][5][1]*0.1973)
    #sigma_in_GeV_err.append(a_mean_err[i][6][1,1]**0.5*0.1973)

ax.errorbar(a_in_fm, sigma_in_GeV, sigma_in_GeV_err, fmt="o")

#plt.legend()
plt.show()
```

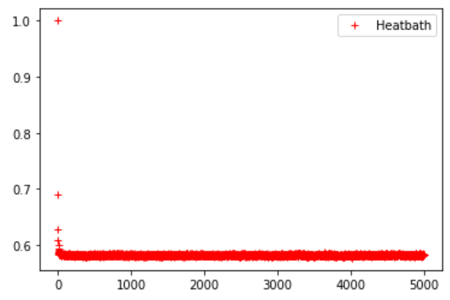


***Homework***: Compute the Coulomb coefficient $B$ in units of GeV fm with $V(r) = A + B/r + \sigma r$ for the same lattice spacings and make a similar plot to the one for $\sigma$.

In [103…
```
def glueball_A1_correlator(U0, nsmear):
    Usmear = g.copy(U0)
    L = U0[0].grid.gdimensions
    for i in range(nsmear):
        Usmear = g.qcd.gauge.smear.stout(Usmear, rho=0.15,orthogonal_dimension=3)
    Pfield1, Pfield2 = g.qcd.gauge.loops.rectangle(Usmear,
                                                   [[ (mu,1,nu,1) for mu in range(3) for nu in range(mu)],
                                                    [ (mu,3,nu,1) for mu in range(3) for nu in range(3) if mu != nu]],
                                                    field=True)
    return (
        [x/(L[0]*L[1]*L[2]) for x in g.slice(Pfield1,3)],
        g.sum(Pfield1) / Pfield1.grid.gsites,
        [x/(L[0]*L[1]*L[2]) for x in g.slice(Pfield2,3)],
        g.sum(Pfield2) / Pfield2.grid.gsites
    )
```

In [81]:
```
#generate_ensemble(8, 32, 5.7, 100, 10000, 5)
generate_ensemble(8, 32, 5.9, 100, 5000, 5)
```

```
GPT :   23458.421117 s : Initializing gpt.random(test,vectorized_ranlux24_24_64) took 3.69549e-05 s
GPT :   23458.458663 s : SU(2)-subgroup heatbath 0 has P = 1.0
GPT :   23792.066852 s : SU(2)-subgroup heatbath 250 has P = 0.5823681093358043
GPT :   24085.247532 s : SU(2)-subgroup heatbath 500 has P = 0.5820270769700736
GPT :   24375.536335 s : SU(2)-subgroup heatbath 750 has P = 0.5827176520994317
GPT :   24667.319774 s : SU(2)-subgroup heatbath 1000 has P = 0.5821944491517359
GPT :   24956.718762 s : SU(2)-subgroup heatbath 1250 has P = 0.5823229107943341
GPT :   25246.901014 s : SU(2)-subgroup heatbath 1500 has P = 0.5830595553523381
GPT :   25535.126178 s : SU(2)-subgroup heatbath 1750 has P = 0.5831393794879535
GPT :   25824.375946 s : SU(2)-subgroup heatbath 2000 has P = 0.5831170585874009
GPT :   26116.473709 s : SU(2)-subgroup heatbath 2250 has P = 0.5807010396911295
GPT :   26403.073038 s : SU(2)-subgroup heatbath 2500 has P = 0.5815773381540428
GPT :   26691.175756 s : SU(2)-subgroup heatbath 2750 has P = 0.5829950881588777
GPT :   26980.989076 s : SU(2)-subgroup heatbath 3000 has P = 0.5817829587379455
GPT :   27274.689568 s : SU(2)-subgroup heatbath 3250 has P = 0.5814936616603346
GPT :   27566.837664 s : SU(2)-subgroup heatbath 3500 has P = 0.5805751461356455
GPT :   27855.261779 s : SU(2)-subgroup heatbath 3750 has P = 0.5800671857797849
GPT :   28138.893678 s : SU(2)-subgroup heatbath 4000 has P = 0.5804503576728808
GPT :   28421.909804 s : SU(2)-subgroup heatbath 4250 has P = 0.5816100159724532
GPT :   28706.761141 s : SU(2)-subgroup heatbath 4500 has P = 0.5817419634089783
GPT :   28989.230684 s : SU(2)-subgroup heatbath 4750 has P = 0.5819070143898305
```



In [108…

```
operators_5p7 = []
for it in range(105,10000,5):
    operators_5p7.append(glueball_A1_correlator(g.load(f"8c32_5.7/su3.{it}"), 6))
    print(it,end="\r")
```

```
9995
```

In [105…

```
operators_5p9 = []
for it in range(105,5000,5):
    operators_5p9.append(glueball_A1_correlator(g.load(f"8c32_5.9/su3.{it}"), 8))
    print(it,end="\r")
```

```
4995
```

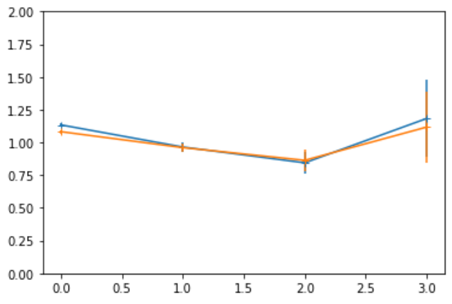
In [111…

```
def bin_data(res, bin_size):
    assert len(res) % bin_size == 0
    n = len(res) // bin_size
    return [ sum(res[bin_size*i:bin_size*(i+1)])/bin_size for i in range(n) ]

def jackknife_variance_estimator(f, x):
    N = len(x)
    X = sum(x) / N
    mean = f(X)
    return sum([(f((N*X - x[j])/(N-1)) - mean)**2 for j in range(N)])*(N-1)/N

def correlate(X, P):
    pdelta = [x - P for x in X]
    Lt = len(pdelta)
    return np.array([ sum([ pdelta[t]*pdelta[(t+dt) % Lt] for t in range(Lt)]) for dt in range(Lt)],dtype=np.complex128)

def glueball_spectrum(operators, a_in_fm):
    plaquette1 = np.mean([op[1] for op in operators])
    plaquette2 = np.mean([op[3] for op in operators])

    correlators1=[correlate(op[0],plaquette1) for op in operators]
    correlators2=[correlate(op[2],plaquette2) for op in operators]

    ldiv8 = len(correlators1)-(len(correlators1)%8)

    err1=[y**0.5 for y in jackknife_variance_estimator(lambda x:x, correlators1)]
    err2=[y**0.5 for y in jackknife_variance_estimator(lambda x:x, bin_data(correlators1[0:ldiv8],4))]
    err4=[y**0.5 for y in jackknife_variance_estimator(lambda x:x, bin_data(correlators1[0:ldiv8],8))]
    print("Binning study of correlator (nbin=1,4,8):",err1[0], err2[0], err4[0])

    corr1=np.mean(correlators1,axis=0)
    emp_mean_1 = [ np.log(corr1[t]/corr1[t+1]) for t in range(4)]
    err_emp_1=[y**0.5 for y in jackknife_variance_estimator(lambda x: np.array([np.log(x[t]/x[t+1]) for t in range(4)]), bin_data(correlators1[

    corr2=np.mean(correlators2,axis=0)
    emp_mean_2 = [ np.log(corr2[t]/corr2[t+1]) for t in range(4)]
    err_emp_2=[y**0.5 for y in jackknife_variance_estimator(lambda x: np.array([np.log(x[t]/x[t+1]) for t in range(4)]), bin_data(correlators2[

    fig, ax = plt.subplots()

    plt.ylim(0,2)
    ax.errorbar(range(len(emp_mean_1)), emp_mean_1, err_emp_1, marker="+")
    ax.errorbar(range(len(emp_mean_2)), emp_mean_2, err_emp_2, marker="+")

    #plt.legend()
    plt.show()

    print("t=1",emp_mean_1[1]*0.1973/a_in_fm,err_emp_1[1]*0.1973/a_in_fm,"GeV")
    print("t=2",emp_mean_1[2]*0.1973/a_in_fm,err_emp_1[2]*0.1973/a_in_fm,"GeV")
```

In [118…

```
print(f"\n beta = {betas[2]}, a = {a_mean_err[2][0]:.2f} fm\n")
glueball_spectrum(operators_5p7, a_mean_err[2][0])

print(f"\n beta = {betas[4]}, a = {a_mean_err[4][0]:.2f} fm\n")
glueball_spectrum(operators_5p9, a_mean_err[4][0])
```

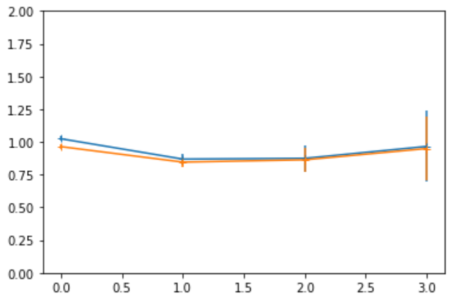
```
 beta = 5.7, a = 0.17 fm

Binning study of correlator (nbin=1,4,8): (9.195666010766062e-07+0j) (1.1338207652267015e-06+0j) (1.235564926411823e-06+0j)
```



```
t=1 (1.1395837688526864+0j) (0.041178362087299636+0j) GeV
t=2 (0.9971292560027318+0j) (0.09740385965874282+0j) GeV
```

```
 beta = 5.9, a = 0.11 fm
```

```
Binning study of correlator (nbin=1,4,8): (2.5400932771942773e-07+0j) (3.226258039722407e-07+0j) (3.490387661363861e-07+0j)
```



```
t=1 (1.5290449848516772+0j) (0.06852290270749113+0j) GeV
t=2 (1.5391866165060522+0j) (0.1773080551731805+0j) GeV
```

***Homework (optional)***: Generate an even finer ensemble with same physical volume as the $\beta = 5.9$ ensemble, determine the lattice spacing and measure the $A_1^{++}$ glueball mass. You should find that once the lattice spacing is sufficiently small, that the glueball state remains at constant energy in GeV (see, e.g., https://doi.org/10.1016/0370-2693(88)91510-9). The $A_1^{++}$ glueball state in quenched QCD is at $1.6(1)$ GeV.