```
# ------------------------------------------------------------------------
#                                Checkerboard
# ------------------------------------------------------------------------
def con_isometry_checkboard(self,l0,angle0):
    """
    keep 2 kinds of diagonal edge-lengths and their crossing angle
    X += [ld1,ld2, ud1,ud2]
    1. (v1-v3) = ld1*ud1, ud1**2=1
    2. (v2-v4) = ld2*ud2, ud2**2=1
    3. ld1 == init_ld1, ld2 == init_ld2
    4. ud1*ud2 == init_ud1*init_ud2
    """
    w = self.get_weight('isometry_checkboard')
    V = self.mesh.V
    num = self.mesh.num_quadface
    N = self.N
    X = self.X
    numl = self._N7-8*num
    numud = self._N7-6*num
    arr = np.arange(num)

    c_ld1 = numl+arr
    c_ld2 = numl+num+arr
    vi = self.mesh.quadface
    v1,v2,v3,v4 = vi[::4],vi[1::4],vi[2::4],vi[3::4]
    c_v1 = np.r_[v1,V+v1,2*V+v1] # [x,y,z]
    c_v2 = np.r_[v2,V+v2,2*V+v2] # [x,y,z]
    c_v3 = np.r_[v3,V+v3,2*V+v3] # [x,y,z]
    c_v4 = np.r_[v4,V+v4,2*V+v4] # [x,y,z]
    c_ud1 = np.r_[numud+arr,numud+num+arr,numud+2*num+arr]
    c_ud2 = c_ud1+3*num

    He1,re1 = self._edge(X,c_v1,c_v3,c_ld1,c_ud1,num,N)
    He2,re2 = self._edge(X,c_v2,c_v4,c_ld2,c_ud2,num,N)
    Hu1,ru1 = self._unit(X,c_ud1,num,N)
    Hu2,ru2 = self._unit(X,c_ud2,num,N)
    Hl1,rl1 = self._constl(c_ld1,l0[:num],num,N)
    Hl2,rl2 = self._constl(c_ld2,l0[num:],num,N)
    Ha,ra = self._constangle(X,c_ud1,c_ud2,angle0,num,N)

    H = sparse.vstack((He1,He2,Hu1,Hu2,Hl1,Hl2,Ha))
    r = np.r_[re1,re2,ru1,ru2,rl1,rl2,ra]
    self.add_iterative_constraint(H*w, r*w, 'isometry(checkboard)')
```

Constraint for isometry_checkerboard way as in Caigui paper

1. basic: get each quad faces' vertex indices
   即 self.mesh.quadface 函数， 见后面
2. 表示isometry条件
   way1: 使用提取给定初始网对角线长度&夹角 (该方法)。
   way2: 使用读取两个对应网格，将其vertices一起作为变量
3. 将下面约束条件表示成稀疏矩阵H 和列表 r
4. 求解非齐次稀疏矩阵线性解

变量X =[vertices, ld1, ld2, ud1, ud2]
Vertices: 所有格点3维坐标
Ld1,ld2: 分别是两组对角线长度
Ud1,ud2：分别是两组对角线方向单位向量

Way1:

$$(v_0 - v_2)^2 = C_0,\ (v_1 - v_3)^2 = C_1,\ (v_0 - v_2)\cdot(v_1 - v_3) = C_3.$$

Way2:

$$c_{iso,0}(f) = (v_0 - v_2)^2 - (v_0' - v_2')^2 = 0,$$
$$c_{iso,1}(f) = (v_1 - v_3)^2 - (v_1' - v_3')^2 = 0,$$
$$c_{iso,2}(f) = (v_0 - v_2)\cdot(v_1 - v_3) - (v_0' - v_2')\cdot(v_1' - v_3') = 0.$$

```python
def _edge(self,X,c_v1,c_v3,c_ld1,c_ud1,num,N):
    "(v1-v3) = ld1*ud1"
    ld1 = X[c_ld1]
    ud1 = X[c_ud1]
    a3 = np.ones(3*num)
    row1 = np.tile(np.arange(3*num),4)
    col = np.r_[c_v1,c_v3,np.tile(c_ld1,3),c_ud1]
    data = np.r_[a3,-a3,-ud1,-np.tile(ld1,3)]
    r = -np.tile(ld1,3)*ud1
    H = sparse.coo_matrix((data,(row1,col)), shape=(3*num, N))
    return H,r

def _unit(self,X,c_ud1,num,N):
    "ud1**2=1"
    arr = np.arange(num)
    row2 = np.tile(arr,3)
    col = c_ud1
    data = 2*X[col]
    r =  np.linalg.norm(X[col].reshape(-1,3,order='F'),axis=1)**2 + np.ones(num)
    H = sparse.coo_matrix((data,(row2,col)), shape=(num, N))
    return H,r

def _constl(self,c_ld1,init_l1,num,N):
    "ld1 == const."
    row3 = np.arange(num,dtype=int)
    col = c_ld1
    data = np.ones(num,dtype=int)
    r = init_l1
    H = sparse.coo_matrix((data,(row3,col)), shape=(num, N))
    return H,r
```

_edge函数
表示：对角线向量==对角线长度*单位向量
被con_isometry_checkberboard函数调用2次
返回稀疏矩阵，和列表 H, r

_unit函数
表示：对角线单位向量
被con_isometry_checkberboard函数调用2次
返回稀疏矩阵，和列表 H, r

_constl函数
表示：对角线长度==给定初始长度值
被con_isometry_checkberboard函数调用2次
返回稀疏矩阵，和列表 H, r

```python
def _constangle(self,X,c_ud1,c_ud2,angle0,num,N):
    "ud1*ud2 == const."
    row4 = np.tile(np.arange(num),6)
    col = np.r_[c_ud1,c_ud2]
    data = np.r_[X[c_ud2],X[c_ud1]]
    r = np.einsum('ij,ij->i',X[c_ud1].reshape(-1,3, order='F'),X[c_ud2].reshape(-1,3,order='F'))+angle0
    H = sparse.coo_matrix((data,(row4,col)), shape=(num, N))
    return H,r
```

_constangle函数
表示：单位对角线向量夹角为给定cos(alpha)
被con_isometry_checkberboard函数调用1次
返回稀疏矩阵，和列表 H, r

```python
def quadfaces(self):
    "for quad diagonals"
    "quadface, num_quadface, quadface_order"
    f, v1, v2 = self.face_edge_vertices_iterators(order=True)
    f4,vi = [],[]
    for i in range(self.F):
        ind = np.where(f==i)[0]
        if len(ind)==4:
            f4.extend([i,i,i,i])
            vi.extend(v1[ind])
            #vj.extend(v2[ind])
    self._num_quadface = len(f4) // 4
    #v1,v2,v3,v4 = vi[::4],vi[1::4],vi[2::4],vi[3::4]
    self._quadface = np.array(vi,dtype=int)
    self._quadface_order = np.unique(f4)
```

由halfedge半边数据结构表示出每个quadface的格点索引值
即返回列表[v1,v2,v3,v4] = quadface

```python
def face_edge_vertices_iterators(self, sort=False, order=False):
    H = self.halfedges
    f  = H[:,1]
    vi = H[:,0]
    vj = H[H[:,2],0]
    if order:
        i  = self.face_ordered_halfedges()
        f  = f[i]
        vi = vi[i]
        vj = vj[i]
    else:
        i  = np.where(H[:,1] >= 0)[0]
        f  = f[i]
        vi = vi[i]
        vj = vj[i]
        if sort:
           i  = np.argsort(f)
           vi = vi[i]
           vj = vj[i]
    return f, vi, vj
```

```python
def face_ordered_halfedges(self):
    H = np.copy(self.halfedges)
    i = np.argsort(H[:,1])
    i = i[np.where(H[i,1] >= 0)]
    f = H[i,1]
    index = np.arange(i.shape[0])
    _, j = np.unique(f, True)
    f = np.delete(f,j)
    index = np.delete(index, j)
    while f.shape[0] > 0:
        _, j = np.unique(f, True)
        i[index[j]] = H[i[index[j] - 1],2]
        f = np.delete(f, j)
        index = np.delete(index, j)
    return i
```