Gradient vector & Hessian matrix - Code

For now let's consider the case without right-censoring.

$$\mu(H_{u_i}, \theta_t) = \theta_0 + \sum_{j=1}^{n-1} \theta_j u_{i-j+1} \tag{1}$$

$$p(u_i|k, \theta_t) = \left[\frac{k}{2\pi \cdot {u_i}^3}\right]^{1/2} e^{-\frac{1}{2}\frac{k\left(u_i - \mu(H_{u_{i-1}}, \theta_t)\right)^2}{\mu(H_{u_{i-1}}, \theta_t)^2 \cdot u_i}} \tag{2}$$

$$\mathcal{L}(u_{t-m:t}|k, \eta_t, \theta_t) = -\sum_{i=0}^{m-1} \eta_i \log p(u_i|k, \theta_t) \tag{3}$$

```
def inverse_gaussian(
    xs: np.array, mus: np.array, lamb: float):
    return np.sqrt(lamb / (2 * np.pi * xs ** 3)) * np.exp(
        (-lamb * (xs - mus) ** 2) / (2 * mus ** 2 * xs)
    )
def compute_invgauss_negloglikel(params: np.array):
    k_param, eta_params, thetap_params = _unpack_invgauss_params(params)
    mus = np.dot(xn, thetap_params)
    # mus.shape : (m,1)
    logps = np.log(inverse_gaussian(wn, mus, k_param))
    return -np.dot(eta_params.T, logps)[0, 0]
```

Gradient `Vector`

$$\nabla \mathcal{L}(u_{t-m:t}|k, \eta_t, \theta_t) = \begin{bmatrix} \frac{\partial}{\partial k}\mathcal{L}(u_{t-m:t}|k, \eta_t, \theta_t) \\ \frac{\partial}{\partial \eta_0}\mathcal{L}(u_{t-m:t}|k, \eta_t, \theta_t) \\ \vdots \\ \frac{\partial}{\partial \eta_{m-1}}\mathcal{L}(u_{t-m:t}|k, \eta_t, \theta_t) \\ \frac{\partial}{\partial \theta_0}\mathcal{L}(u_{t-m:t}|k, \eta_t, \theta_t) \\ \vdots \\ \frac{\partial}{\partial \theta_{n-1}}\mathcal{L}(u_{t-m:t}|k, \eta_t, \theta_t) \end{bmatrix} \tag{4}$$

Where:

$$
\begin{aligned}
\frac{\partial}{\partial k}\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) &= \sum_{i=0}^{m-1}\frac{\eta_i}{2}\left(-\frac{1}{k}+\frac{(u_i-\mu(H_{u_{i-1}},\theta_t))^2}{\mu(H_{u_{i-1}},\theta_t)^2\cdot u_i}\right) \\
\frac{\partial}{\partial \eta_j}\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) &= -\log p(u_j|k,\theta_t) \\
\frac{\partial}{\partial \theta_j}\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) &= -\sum_{i=0}^{m-1}\eta_i k\cdot\frac{(u_i-\mu(H_{u_{i-1}},\theta_t))\,(u_{i-j})}{\mu(H_{u_{i-1}},\theta_t)^3}
\end{aligned}
\tag{5}
$$

```
def compute_invgauss_negloglikel_grad(params: np.array):
    """
    returns the vector of the first-derivatives of the negloglikelihood w.r.t
to each        parameter
    """
    # Retrieve the useful variables
    k_param, eta_params, thetap_params = _unpack_invgauss_params(params)
    mus = np.dot(xn, thetap_params).reshape((m,1))

    # Compute the gradient for k
    tmp = -1/k + (wn-mus)**2/(mus**2*wn)
    k_grad = np.dot((eta_params/2).T,tmp)

    # Compute the gradient for eta[0]...eta[m-1]
    eta_grad = -1*np.log(inverse_gaussian(wn,mus,k_param))

    # Compute the gradient form thetap[0]...thetap[n-1]
    tmp = -1*k_param*eta_params*(wn-mus)/mus**3
    thetap_grad = np.dot(tmp.T,xn).T

    # Return all the gradients as a single vector of shape (n+m+1,)
    return np.vstack([k_grad,eta_grad,thetap_grad]).squeeze(1)
```

Hessian `Matrix`

$$
\nabla^2\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) =
\begin{bmatrix}
\frac{\partial^2}{\partial^2 k}\mathcal{L}(\cdot) & \frac{\partial}{\partial k\partial\eta_0}\mathcal{L}(\cdot) & \cdots & \frac{\partial}{\partial k\partial\eta_{m-1}}\mathcal{L}(\cdot) & \frac{\partial}{\partial k\partial\theta_0}\mathcal{L}(\cdot) & \cdots & \frac{\partial}{\partial k\partial\theta_{n-1}}\mathcal{L}(\cdot) \\
\cdot & \frac{\partial^2}{\partial^2 \eta_0}\mathcal{L}(\cdot) & \cdots & \frac{\partial}{\partial \eta_0\partial\eta_{m-1}}\mathcal{L}(\cdot) & \frac{\partial}{\partial \eta_0\partial\theta_0}\mathcal{L}(\cdot) & \cdots & \frac{\partial}{\partial \eta_0\partial\theta_{n-1}}\mathcal{L}(\cdot) \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
\cdot & \cdot & \cdots & \frac{\partial^2}{\partial^2 \eta_{m-1}}\mathcal{L}(\cdot) & \frac{\partial}{\partial \eta_{m-1}\partial\theta_0}\mathcal{L}(\cdot) & \cdots & \frac{\partial}{\partial \eta_{m-1}\partial\theta_{n-1}}\mathcal{L}(\cdot) \\
\cdot & \cdot & \cdots & \cdot & \frac{\partial^2}{\partial^2 \theta_0}\mathcal{L}(\cdot) & \cdots & \frac{\partial}{\partial \theta_0\theta_{n-1}}\mathcal{L}(\cdot) \\
\cdot & \cdot & \vdots & \cdot & \cdot & \ddots & \vdots \\
\cdot & \cdot & \cdots & \cdot & \cdot & \cdots & \frac{\partial^2}{\partial^2 \theta_{n-1}}\mathcal{L}(\cdot)
\end{bmatrix}
\tag{6}
$$

Where:

```python
for i in range(m+1,m+n+1):
    for j in range(m+1+i,m+1+n):
        tmp1 = xn[:,i-m-1]*xn[:,j-m-1]
        tmp2 = eta_params*k_param*(3*wn-2*mus)/(mus**4)
        hess[i,j] = np.dot(tmp1.T,tmp2)
```

$$\frac{\partial^2}{\partial^2 k}\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) = \sum_{i=0}^{m-1}\frac{\eta_i}{2}k^{-2} \tag{7}$$

$$\frac{\partial}{\partial k\partial\eta_j}\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) = \frac{1}{2}\left(-\frac{1}{k}+\frac{\left(u_j-\mu(H_{u_{j-1}},\theta_t)\right)^2}{\mu(H_{u_{j-1}},\theta_t)^2\cdot u_j}\right) \tag{8}$$

$$\frac{\partial}{\partial k\partial\theta_j}\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) = -\sum_{i=0}^{m-1}\eta_i\cdot\frac{\left(u_i-\mu(H_{u_{i-1}},\theta_t)\right)(u_{i-j})}{\mu(H_{u_{i-1}},\theta_t)^3} \tag{15}$$

$$\frac{\partial^2}{\partial^2\eta_j}\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) = 0 \tag{10}$$

$$\frac{\partial}{\partial\eta_j\eta_q}\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) = 0 \tag{11}$$

$$\frac{\partial}{\partial\eta_j\theta_q}\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) = -k\cdot\frac{\left(u_j-\mu(H_{u_{j-1}},\theta_t)\right)(u_{j-q})}{\mu(H_{u_{j-1}},\theta_t)^3} \tag{12}$$

$$\frac{\partial^2}{\partial^2\theta_j}\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) = \sum_{i=0}^{m-1}\eta_i k(u_{i-j})^2\cdot\left(\frac{3\cdot u_i-2\cdot\mu(H_{u_{i-1}},\theta_t)}{\mu(H_{u_{i-1}},\theta_t)^4}\right) \tag{13}$$

$$\frac{\partial}{\partial\theta_j\partial\theta_q}\mathcal{L}(u_{t-m:t}|k,\eta_t,\theta_t) = \sum_{i=0}^{m-1}\eta_i k(u_{i-j})(u_{i-q})\cdot\left(\frac{3\cdot u_i-2\cdot\mu(H_{u_{i-1}},\theta_t)}{\mu(H_{u_{i-1}},\theta_t)^4}\right) \tag{14}$$

```python
def compute_invgauss_negloglikel_hessian(params: np.array):
    """
    returns the vector of the second-derivatives of the negloglikelihood w.r.t
to each
    parameter
    """
    # Retrieve the useful variables
    k_param, eta_params, thetap_params = _unpack_invgauss_params(params)
    mus = np.dot(xn, thetap_params).reshape((m, 1))

    # Initialize hessian matrix
    hess = np.zeros((1+m+n,1+m+n))

    # We populate the hessian as a upper triangular matrix
    # by filling the rows starting from the main diagonal

    # Partial derivatives w.r.t. k
    kk = np.sum(eta_params)*1/(2*k**2)
```

```
keta = 1/2*(-1/k + (wn-mus)**2/(mus**2*wn)).squeeze(1)
tmp = eta*(wn-mus)/mus**3
ktheta = -np.dot(tmp.T, xn).T.squeeze(1)

hess[0,0] = kk
hess[0, 1:(1+m)] = keta
hess[0, (1+m):(1+m+n)] = ktheta

# All the partial derivatives in the form eta_j\eta_q are null
for i in range(1,m+1):
    for j in range(i,m+1):
        hess[i,j] = 0

#TODO is there a smarter way? (eta_j\theta_q)
for i in range(1,m+1):
    for j in range(m+1+i,m+1+n):
        hess[i,j] = -k*(xn[i-1,j-m-1])*(wn[i-1]-mus[i-1])/mus[i-1]**3

#TODO is there a smarter way? (theta_j\theta_q)
for i in range(m+1,m+n+1):
    for j in range(m+1+i,m+1+n):
        tmp1 = xn[:,i-m-1]*xn[:,j-m-1]
        tmp2 = eta_params*k_param*(3*wn-2*mus)/(mus**4)
        hess[i,j] = np.dot(tmp1.T,tmp2)


# Populate the rest of the matrix
hess = np.where(hess,hess,hess.T)
return hess
```