SR1 update: computes $\frac{(y-B^k s)(y-B^k s)^T}{(y-B^k s)^T s}$ in SR1 algorithm

Input:

- change in x, s
- change in gradient, y
- iteration no., k
- Hessian approximation, B

* compute repetitive component of SR1 update

$u = y - B^{k-1} \circ s$

* compute denominator

$\text{denom} = u \circ s$

* check that denominator is bounded away from zero, i.e., $\|\text{denom}\| \geq c_1 \|s\|^2$

if $|\text{denom}| \leq 1E\text{-}10$:

  * skip update

  $dB = 0$

else:

  * calculate update

  $dB = \frac{u \circ u^T}{\text{denom}}$

Output:

- dB, SR1 update

BFGS Update:

Input:
- change in x, s
- change in gradient, y
- iteration number, k
- Hessian approximation, B

```
* compute denominator of term 1
denom = s^T y

* compute term 1
term 1 = y · y^T / denom

* compute term 2
term 2 = B^k s s^T B^k / (s^T B^k s)

* ensure s^T y is sufficiently positive
c2 = 1E-4
  if denom ≤ c2 :
     * skip update
     dB = 0
  else
     * update
     dB = term1 - term2
```

$$\text{term 2} = \frac{B^k s s^T B^k}{s^T B^k s}$$

Output:
- dB, BFGS update

Line Search:

Inputs:
- decision variables, x
- objective values, f
- gradient, grad
- function to minimize, calc_f
- step, p_k
- iteration no., k
- initial step length for scaling for line search, α_{max}
- Goldstein-Armijo conditions (line search), η_{ls}
- parameter to shrink, ρ_{ls}

```
* indicate line search
ls = True

*initialize α
α^k = α_max

while ls:

    * compute test point
    xtest = x(k) + α^k p^k

    *compute function at test point
    ftest = calc_f(xtest)

    * compute RHS of equation 3.31
    rhs = f^k + η_ls α^k grad^T · p^k

    * check satisfaction of 3.31
    if f_test > rhs:
        * update line search
        α^k = ρ_ls α^k
    else:
        * 3.31 satisfied, break while loop
        ls = False

*update line search
update = α^k p^k
```

Output:
- update
- α^k

Trust Region:

Inputs:

- decision variables, x
- gradients, grad
- Hessian approximation, B
- trust region size, delta
- iteration number, k
- step, p^k
- initial trust region size, delta_0_tr

```
## initialize trust region radius
if k=0:
   delta.append(delta_0_tr)

grad-zero = min(norm(grad[k]), 10^-14)

## Powell dogleg step

# Calculate Cauchy step, pC

denom = grad[k] ∘ B[k] ∘ grad[k]

if denom > 0:

     pc = (-grad[k] ∘ grad[k])/denom ∘ grad[k]

else:

     pc = -delta/norm(grad[k]) ∘ grad[k]

# get pN

pN = pK

# solve equation 3.49

if delta => norm(pN):

   update = pN

elif delta <= norm(pc):

   update = delta∘pc /norm(pC)

else:

   term1 = (pN-pC)^T pc
   term2 = ((pN-pC)^T pC)^2 + delta^2 - norm(pc)^2 norm(pN-pC)^2
   term3 = norm(pN-pC)^2
   n = (term1 + sqrt(term2)) / term3
   update = npN + (1-n)pC

return:
update
```