CS205L Homework9

Philippe Weingertner (pweinger@stanford.edu)

March 12, 2020

This homework was done by:

- Name: Philippe Weingertner (pweinger@stanford.edu)
- SUID: 06325134

Without collaborators.

1 Problem 9.1 (*) Quasi-Newton methods: The Good Broyden's Method

1. $P = \left(A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}\right)\left(A + uv^T\right) = I + A^{-1}uv^T - \frac{A^{-1}uv^T + A^{-1}uv^TA^{-1}uv^T}{1+v^TA^{-1}u}$
   but $v^TA^{-1}u$ is a scalar
   So we can factor out $v^TA^{-1}u$ in $A^{-1}uv^TA^{-1}uv^T$ and get
   $P = I + A^{-1}uv^T - \frac{\left(1+v^TA^{-1}u\right)\left(A^{-1}uv^T\right)}{1+v^TA^{-1}u} = I$
   So the formula is verified
   $$\left(A + uv^T\right)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}$$
2. $B_{k+1} = B_k + \left(y_k - B_ks_k\right)\frac{s_k^T}{s_k^Ts_k}$

   We set $A = B_k, u = y_k - B_ks_k, v^T = \frac{s_k^T}{s_k^Ts_k}$

   And get: $B_{k+1}^{-1} = B_k^{-1} - \frac{B_k^{-1}(y_k - B_ks_k)\frac{s_k^T}{s_k^Ts_k}B_k^{-1}}{1+\frac{s_k^T}{s_k^Ts_k}B_k^{-1}(y_k - B_ks_k)} = B_k^{-1} - \frac{B_k^{-1}(y_k-B_ks_k)s_k^TB_k^{-1}}{s_k^TB_k^{-1}y_k}$
   $$B_{k+1}^{-1} = B_k^{-1} + \frac{s_k - B_k^{-1}y_k}{s_k^TB_k^{-1}y_k}s_k^TB_k^{-1}$$
3. We replace the step: Solve $B_ks_k = -f\left(x_k\right)$ wich is $s_k = -B_k^{-1}f\left(x_k\right)$

   with: $s_k = -B_k^{-1}f\left(x_k\right) = -\left(B_{k-1}^{-1} + \frac{s_{k-1} - B_{k-1}^{-1}y_{k-1}}{s_{k-1}^TB_{k-1}^{-1}y_{k-1}}s_{k-1}^TB_{k-1}^{-1}\right)f\left(x_k\right)$

So we are left with only matrix and vector multiplication to update directly the inverse matrix B_k^{-1} based on previous inverse matrix B_{k-1}^{-1}

And then multiply this matrix with the vector $f(x_k)$

4. To compute B_k^{-1} we have to:
   - update a column vector: $s_{k-1} - B_{k-1}^{-1} y_{k-1}$ via matrix-vector multiplication and vector-vector substraction
   - update a row vector: $s_{k-1}^T B_{k-1}^{-1}$ via vector-matrix multiplication
   - peform an outer product with the 2 above vectors
   - compute a scalar product: $\left(s_{k-1}^T B_{k-1}^{-1}\right) y_{k-1}$
   - Add previous inverse matrix with outer product result (scaled by a scalar)

   In terms of complexity we are down from $\mathcal{O}\left(n^3\right)$ for solving a linear system to $\mathcal{O}\left(n^2\right)$ dealing with a few simple matrix-vector multiplications.

   Moreover with today's technology, GPU and multicores, matrix-matrix or matrix-vector operations are parallelized $AB = [c_{ij}]$ with $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ can be computed in parallel.

2 Problem 9.2 (***) Quasi-Newton Methods for Optimization - BFGS

1. $H_{k+1} = H_k + \frac{y_k y_k^T}{y_k^T x_k} - \frac{H_k x_k x_k^T H_k^T}{x_k^T H_k x_k}$

   $H_{k+1} x_k = H_k x_k + \frac{y_k y_k^T x_k}{y_k^T y_k} - \frac{H_k x_k x_k^T H_k^T x_k}{x_k^T H_k x_k}$

   We note that $x_k^T H_k x_k$ is a scalar and then $x_k^T H_k x_k = \left(x_k^T H_k x_k\right)^T = x_k^T H_k^T x_k$

   $H_{k+1} x_k = H_k x_k + \frac{y_k y_k^T x_k}{y_k^T x_k} - H_k x_k = y_k$

2. Symmetry if H_k is symmetric:
   - $y_k y_k^T$ is symmetric
   - $y_k^T x_k$ and $x_k^T H_k x_k$ are scalars and hence symmetric
   - $H_k x_k x_k^T H_k^T = \left(H_k x_k x_k^T H_k^T\right)^T$ is symmetric

   So if H_k is symmetric then $H_k + \frac{y_k y_k^T}{y_k^T x_k} - \frac{H_k x_k x_k^T H_k^T}{x_k^T H_k x_k}$ is symmetric and hence H_{k+1} is symmetric

3. Positive Definiteness

   Starting with Cauchy-Schwarz inequality as H is SPD, we have: $\left(v^T H x\right)^2 \leq \left(v^T H v\right)\left(x^T H x\right)$

But $v^T Hx$ is a scalar so $\left(v^T Hx\right)^2 = v^T Hxx^T H^T v$ and we get

$v^T Hxx^T H^T v \leq \left(v^T Hv\right)\left(x^T Hx\right)$

H is SPD so $x^T Hx > 0$ for $x \neq 0$ and then:

$\frac{v^T Hxx^T H^T v}{x^T Hx} \leq v^T Hv$ and then

$v^T \left(H - \frac{Hxx^T H^T}{x^T Hx}\right) v \geq 0$ se we have 1st term + 3rd term ≥ 0

For the 2nd term: $v^T \left(\frac{yy^T}{y^T x}\right) v = \frac{\left(v^T y\right)\left(y^T v\right)}{y^T x} = \frac{\left(y^T v\right)^2}{y^T x} \geq 0$ as $y^T x > 0$ enforced in line search

Now, if 1st term + 3rd term = 0, then from Cauchy-Schwarz iniquality we have $v = \lambda x$ and for the 2nd term we have:

$v^T \left(\frac{yy^T}{y^T x}\right) v = \frac{\left(v^T y\right)\left(y^T v\right)}{y^T x} = \lambda^2 \frac{\left(x^T y\right)\left(y^T x\right)}{y^T x} = \lambda^2 \left(y^T x\right) > 0$ as $\lambda \neq 0$ and $y^T x > 0$

So we can summarize: if H_k is SPD then $v^T H_{k+1} v \geq 0$ as it is the sum of 3 terms ≥ 0

And $v^T H_{k+1} v = 0$ if (1st term + 3rd term)=0 **and** 2nd term=0

But if (1st term + 3rd term)=0 then 2nd term > 0 for $v \neq 0$

So $v^T H_{k+1} v = 0$ implies $v = 0$

And we conclude, if H_k is SPD then H_{k+1} is SPD: $v^T H_{k+1} v > 0$ for $v \neq 0$

4. Inverse

   $H_{k+1} = H_k + \frac{y_k y_k^T}{y_k^T x_k} - \frac{H_k x_k x_k^T H_k^T}{x_k^T H_k x_k} = H_k + \gamma uu^T + \delta ww^T$

   with $\gamma = \frac{1}{y_k^T x_k}, u = y_k, \delta = -\frac{1}{x_k^T H_k x_k}, w = H_k x_k$

   And from there we can apply Sherman-Morisson formula like in problem 9.1

3 Problem 9.3 (**) Evaluating Gradient Descent Using Condition Number

1. $x^{k+1} = x^k - \alpha \nabla f\left(x_k\right)$ and $\nabla f\left(x\right) = Ax - b$ and we assume A is SPD

   (a) With $A = Q\Sigma Q^T$ we get $x^{k+1} = x^k - \alpha\left(Q\Sigma Q^T x^k - b\right)$

   (b) For the minimum we have $Q\Sigma Q^T x^* = b$ so we can write
   $x^{k+1} = x^k - \alpha Q\Sigma Q^T\left(x^k - x^*\right) = x^k - \alpha Q\Sigma z^k$
   $x^{k+1} - x^* = \left(x^k - x^*\right) - \alpha Q\Sigma z^k$
   $Q^T\left(x^{k+1} - x^*\right) = Q^T\left(x^k - x^*\right) - \alpha\Sigma z^k$
   $z^{k+1} = z^k - \alpha\Sigma z^k = \left(I - \alpha\Sigma\right) z^k$

$$z^{k+1} = \left(I - \alpha\Sigma\right) z^k$$

So we have

$$z_i^{k+1} = (1-\alpha\lambda_i)\, z_i^k$$

(c) $z_i^k = (1-\alpha\lambda_i)\, z_i^{k-1}$
$z_i^{k-1} = (1-\alpha\lambda_i)\, z_i^{k-2}$
$\vdots$
$z_i^1 = (1-\alpha\lambda_i)\, z_i^0$
So we get:

$$z_i^k = (1-\alpha\lambda_i)^k\, z_i^0$$

(d) $Q^T\left(x_i^k - x_i^*\right) = (1-\alpha\lambda_i)^k\, Q^T\left(x_i^0 - x_i^*\right)$
$e_i^k = (1-\alpha\lambda_i)^k\, e_i^0$

$$e^k = \begin{bmatrix} (1-\alpha\lambda_1)^k & & \\ & \ddots & \\ & & (1-\alpha\lambda_n)^k \end{bmatrix} e^0$$

2. $|1-\alpha\lambda_i| < 1 \Longleftrightarrow 0 < \alpha\lambda_i < 2$

3. $\min\limits_{\alpha} \text{rate}(\alpha) = \frac{\lambda_n-\lambda_1}{\lambda_n+\lambda_1} \geq 0$

   (a) So the values of λ_1, λ_n that minimize this equation are such that $\lambda_n = \lambda_1$ which corresponds to an optimal condition number of 1

   (b) $\min\limits_{\alpha} \text{rate}(\alpha) = \frac{\lambda_n-\lambda_1}{\lambda_n+\lambda_1} = \frac{\frac{\lambda_n}{\lambda_1}-1}{\frac{\lambda_n}{\lambda_1}+1} = \frac{\kappa-1}{\kappa+1}$ (The matrix is SPD so all eigenvalues are $\lambda_i > 0$)

4 Problem 9.5 (*) Stability and Error of ODEs

$\mathbf{c}' = e^{-t}A\mathbf{c}$ in the form of $c'(t) = f(t, c(t))$

1. Forward Euler: $c_{n+1} = c_n + hc'_n$ i.e. $\mathbf{c_{n+1}} = \mathbf{c_n} + he^{-t_n}A\mathbf{c}_n$

   Backward Euler: $c_{n+1} = c_n + hc'_{n+1}$ i.e. $\mathbf{c_{n+1}} = \mathbf{c_n} + he^{-t_{n+1}}A\mathbf{c_{n+1}}$

   Trapezoidal rule: $c_{n+1} = c_n + \frac{h}{2}\left(c'_n + c'_{n+1}\right)$ i.e. $\mathbf{c_{n+1}} = \mathbf{c_n} + \frac{h}{2}\left(e^{-t_n}A\mathbf{c}_n + e^{-t_{n+1}}A\mathbf{c}_{n+1}\right)$

2. Forward Euler: ++ Simple to compute (no matrix inversion), — To be stable requires small step size

   Backward Euler: the opposite i.e. ++ Is unconditionally stable — Requires solving a Linear System

   Trapezoidal rule: ++ Is unconditionally stabel, — Requires solving a Linear System, can have unwanted oscillations

5 Problem 9.6 (*) A Review on Optimizers

We have $L : \mathbb{R}^2 \to \mathbb{R}$ with $L(w) = \sum_i (w_1 x_i + w_2 - y_i)^2$

1. $\frac{\partial L}{\partial w_1} = \sum_i 2(w_1 x_i + w_2 - y_i) x_i$

   $\frac{\partial L}{\partial w_2} = \sum_i 2(w_1 x_i + w_2 - y_i)$

   $$\begin{bmatrix} w_{t+1,1} \\ w_{t+1,2} \end{bmatrix} = \begin{bmatrix} w_{t,1} \\ w_{t,2} \end{bmatrix} - \eta \begin{bmatrix} \sum_i 2(w_{t,1} x_i + w_{t,2} - y_i) x_i \\ \sum_i 2(w_{t,1} x_i + w_{t,2} - y_i) \end{bmatrix}$$

2. With momentum, as per programming assignment 8:

   $$\begin{cases} v_{t+1} = \gamma v_t + \eta \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \end{bmatrix} \\ \begin{bmatrix} w_{t+1,1} \\ w_{t+1,2} \end{bmatrix} = \begin{bmatrix} w_{t,1} \\ w_{t,2} \end{bmatrix} - v_{t+1} \end{cases}$$

   Another way to look at it could be:

   $$\begin{cases} v_{t+1} = \gamma v_t + (1-\gamma) \nabla L & \text{smoothing/weighted average or L1-filtering of gradient} \\ \begin{bmatrix} w_{t+1,1} \\ w_{t+1,2} \end{bmatrix} = \begin{bmatrix} w_{t,1} \\ w_{t,2} \end{bmatrix} - \alpha v_{t+1} \end{cases}$$

3. With Adagrad:

   $w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{\sum_t g_{t,i}^2 + \epsilon}} g_{t,i}$ with $g_{t,i} = \frac{\partial L(w)}{\partial w_i}$

   If we consider a dataset where some features are very frequent and others are less frequent:

   - Features less frequent: we want to use higher Learning Rate
   - Features more frequent: we want to use lower Learning Rate

   Otherwise if we use the same LR for all features, the sparse/rare features will have very limited impact on the training and update rule, not because they are not important, but just because they are encountered less often. So to avoid this problem we want to weight more the less frequent features when we encounter them. This is what we are doing via $\frac{1}{\sqrt{\sum_t g_{t,i}^2}}$: we are summing over time. The more frequent features will have a bigger $G_{t,i} = \sum_t g_{t,i}^2$ and hence a smaller update weight, while the less frequent features will have a smaller $G_{t,i}$ and hence a bigger update weight when we encounter them.

   But as seen in Unit 20, the learning rates monotonically decrease and often go to zero too quickly stalling out the algorithm.

6 Problem 9.4 (*) k-th order Runge Kutta

1. From discrete: $c(t_{k+1}) = c(t_k) - \alpha \nabla_c F(c(t_k))$ which is Forward Euler with step size α

   To continuous formulation: $\frac{dc}{dt}(t) = -\nabla F(c(t))$

   $$\begin{bmatrix} \frac{dc_1}{dt}(t) \\ \frac{dc_2}{dt}(t) \end{bmatrix} = -\begin{bmatrix} \frac{1}{4}(c_1(t) - 4) \\ 4\left(c_2(t) - \frac{1}{4}\right) \end{bmatrix} = \begin{bmatrix} 1 - \frac{1}{4}c_1(t) \\ 1 - 4c_2(t) \end{bmatrix}$$

   We get $\begin{cases} 4c_1'(t) = 4 - c_1(t) \\ c_2'(t) = 1 - 4c_2(t) \end{cases} \iff \begin{cases} c_1(t) = K_1 e^{\frac{-t}{4}} + 4 \\ c_2(t) = K_2 e^{-4t} + \frac{1}{4} \end{cases}$

   If we start with $c(t=0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ we have $\begin{cases} K_1 = -4 \\ K_2 = -\frac{1}{4} \end{cases}$

   We get: $\begin{cases} c_1(t) = 4\left(1 - e^{-\frac{t}{4}}\right) \\ c_2(t) = \frac{1}{4}\left(1 - e^{-4t}\right) \end{cases}$ and $\begin{cases} c_1\left(\frac{1}{2}\right) = 4\left(1 - e^{-\frac{1}{8}}\right) = 0.47 \\ c_2\left(\frac{1}{2}\right) = \frac{1}{4}\left(1 - e^{-2}\right) = 0.2162 \end{cases}$

2. Evaluate RK1,2,4: $f\left(t_n, \begin{bmatrix} c_{1,n} \\ c_{2,n} \end{bmatrix}\right) = \begin{bmatrix} 1 - \frac{1}{4}c_{1,n} \\ 1 - 4c_{2,n} \end{bmatrix}$

   - With RK1:

     $c(t_{k+1}) = c(t_k) + hc'(t_k)$ so we get $\begin{bmatrix} c_1(t_1) \\ c_2(t_1) \end{bmatrix} = \begin{bmatrix} c_1(t_0) \\ c_2(t_0) \end{bmatrix} + h\begin{bmatrix} 1 - \frac{1}{4}c_1(t_0) \\ 1 - 4c_2(t_0) \end{bmatrix}$

     $\begin{bmatrix} c_1(t_1) \\ c_2(t_1) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 - \frac{1}{4}c_1(t_0) \\ 1 - 4c_2(t_0) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$

   - With RK2:

     $k_1 = f\left(t_0, \begin{bmatrix} c_{1,0} \\ c_{2,0} \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, k_2 = f\left(t_0 + h, \begin{bmatrix} c_{1,0} \\ c_{2,0} \end{bmatrix} + hk_1\right) = f\left(\frac{1}{2}, \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}\right) = \begin{bmatrix} 1 - \frac{1}{8} \\ 1 - 2 \end{bmatrix} = \begin{bmatrix} \frac{7}{8} \\ -1 \end{bmatrix}$

     $\begin{bmatrix} c_{1,1} \\ c_{2,1} \end{bmatrix} = \begin{bmatrix} c_{1,0} \\ c_{2,0} \end{bmatrix} + h\left(\frac{k_1}{2} + \frac{k_2}{2}\right) = \frac{1}{4}\begin{bmatrix} \frac{15}{8} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{15}{32} \\ 0 \end{bmatrix} = \begin{bmatrix} 0.4688 \\ 0 \end{bmatrix}$

   - With RK4: as a reminder $f\left(t_n, \begin{bmatrix} c_{1,n} \\ c_{2,n} \end{bmatrix}\right) = \begin{bmatrix} 1 - \frac{1}{4}c_{1,n} \\ 1 - 4c_{2,n} \end{bmatrix}$ and $\begin{bmatrix} c_{1,0} \\ c_{2,0} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

     $k_1 = f\left(t_0, \begin{bmatrix} c_{1,0} \\ c_{2,0} \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, k_2 = f\left(t_0 + \frac{h}{2}, \begin{bmatrix} c_{1,0} \\ c_{2,0} \end{bmatrix} + \frac{h}{2}k_1\right) = f\left(\frac{1}{4}, \begin{bmatrix} \frac{1}{4} \\ \frac{1}{4} \end{bmatrix}\right) = \begin{bmatrix} \frac{15}{16} \\ 0 \end{bmatrix}$

     $k_3 = f\left(t_0 + \frac{h}{2}, \begin{bmatrix} c_{1,0} \\ c_{2,0} \end{bmatrix} + \frac{h}{2}k_2\right) = f\left(\frac{1}{4}, \begin{bmatrix} \frac{15}{64} \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 1 - \frac{15}{256} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{241}{256} \\ 1 \end{bmatrix}$

$$k_4 = f\left(t_0 + h, \begin{bmatrix} c_{1,0} \\ c_{2,0} \end{bmatrix} + hk_3\right) = f\left(\frac{1}{2}, \begin{bmatrix} \frac{241}{512} \\ \frac{1}{2} \end{bmatrix}\right) = \begin{bmatrix} 1 - \frac{1}{4}\frac{241}{512} \\ 1 - 4\frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{2048-241}{2048} \\ -1 \end{bmatrix} = \begin{bmatrix} \frac{1807}{2048} \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} c_{1,n} \\ c_{2,n} \end{bmatrix} = \begin{bmatrix} c_{1,0} \\ c_{2,0} \end{bmatrix} + h\left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right) = \frac{1}{2}\left(\frac{1}{6}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \frac{1}{3}\begin{bmatrix} \frac{15}{16} \\ 0 \end{bmatrix} + \frac{1}{3}\begin{bmatrix} \frac{241}{256} \\ 1 \end{bmatrix} + \frac{1}{6}\begin{bmatrix} \frac{1807}{2048} \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 0.47 \\ 0.1667 \end{bmatrix}$$

```python
import math
import numpy as np

# c'(t)=f(t,c(t)) with c,c' vectors
def f(t, c):
  c1, c2 = c[0], c[1]
  y1 = 1-0.25*c1
  y2 = 1-4*c2
  return np.array([y1,y2])

def solf(t):
  y1 = 4*(1-math.exp(-t/4))
  y2 = 0.25*(1-math.exp(-4*t))
  return np.array([y1,y2])

def rk1(c, t, h, f):
  k1 = f(t,c) # vector
  cnext = c + h*k1
  tnext = t + h
  return cnext, tnext # vector, real

def rk2(c, t, h, f):
  k1 = f(t,c) # vector
  k2 = f(t+h, c+h*k1)
  cnext = c + h*(k1/2+k2/2)
  tnext = t + h
  return cnext, tnext # vector, real

def rk4(c, t, h, f):
  k1 = f(t,c) # vector
  k2 = f(t+h/2, c+h/2*k1)
  k3 = f(t+h/2, c+h/2*k2)
  k4 = f(t+h, c+h*k3)
  cnext = c + h*(k1/6+k2/3+k3/3+k4/6)
  tnext = t + h
  return cnext, tnext # vector, real

def solver_rk(c, t, h, steps):
  ts, cs, cexacts = [t], [c], [solf(t)]
  print("Runge Kutta method with h={}".format(h))
  for k in range(steps):
    c, t = rk4(c, t, h, f)
    cexact = solf(t)
    err = np.linalg.norm(c-cexact)
    print("c({:.2f})={} with err={}".format(t, c, err))
    ts.append(t)
    cs.append(c)
    cexacts.append(cexact)
  return ts, cs, cexacts

c0=np.zeros((2,))
t0, h, steps = 0, 0.5, 1
ts, cs, cexacts = solver_rk(c0, t0, h, steps)
print("c exact: {}".format(cexacts))
print("c: {}".format(cs))
```