REPUBLIQUE TUNISIENNE

Ministère de l'Enseignement Supérieur,
de la Recherche Scientifique

Concours Nationaux d'Entrée
aux Cycles de Formation d'Ingénieurs
Session 2020

الجمهورية التونسية
وزارة التعليم العالي والبحث العلمي

المناظرات الوطنية للدخول
إلى مراحل تكوين المهندسين
دورة 2020

Alternative de Correction Concours Mathématiques et Physique, Physique et Chimie et Technologie
Epreuve d'Informatique

Barème sur 100
PROBLEME 1 (65 pts)
Partie 1 :

1. **1.25 pt**

Version 1:

```
Tinitial= lambda x: 0 if x==0 or x==L else 200
```

Version 2:

```
def Tinitial(x):
    return 0 if x in (0,L) else 200
```

Version 3:

```
def Tinitial(x):
   assert 0<=x<=L , 'longueur depassant la tige'
   if x==0 or x==L :
      return 0
   else:
      return 200
```

2. **1.25 pt**

Version 1:

```
def Fn(x,n):
    return Tinitial(x) * np.sin((n*np.pi*x)/L)
```

Version 2:

```
Fn=lambda x,n:Tinitial(x)*np.sin((np.pi*n*x)/L)
```

3. **2.5 pts**

```
def Dn(n):
    Q=spi.quad(Fn,0,L,(n,))
    return (2/L)*Q[0]
```

4. **5 pts**

```
def SolutionAnalytique(x,t,eps):
     n=1
     s=0
     while True:
        t=Dn(n)*np.sin(n*x*np.pi/L)*np.exp((-n*n*np.pi**2*alpha*t)/L**2)
        s+=t
        n+=1
```

```
    if abs(t)< eps:
          return s
```

Partie 2

5. 5 pts

Version 1:

```
def GenererA(N):
     A=np.zeros((N+1,N+1),'int')
     for i in range(1,N+1):
          A[i,i]=-2
          A[i-1,i]=1
          A[i,i-1]=1
     A[0]=A[N]=np.zeros(N+1)
     return A
```

Version 2:

```
def GenererA(N):
    a=np.diag([-2 for i in range(0,N+1)])
    b=np.diag([1 for i in range(0,N)],1)
    c=np.diag([1 for i in range(0,N)],-1)
    A=a+b+c
    A[0,:]=A[N,:]=0
    return A
```

Version 3:

```
def GenererA(N):
    A=np.zeros((N+1,N+1))
    for i in range(1,N):
        for j in range(N+1):
            if i==j :
                A[i,j]=-2
            elif i==j+1 or i==j-1:
                A[i,j]=1
    return A
```

Version 4:

```
def GenererA(N):
    L=[-2 for i in range(N+1)]
    A=np.diag(L)
    A[0,0]=A[N,N]=0
    for i in range(1,N):
        for j in range(N+1):
            if i==j+1 or i==j-1:
                A[i,j]=1
    return A
```

Version 5:

```
def GenererA(N):
      fill = lambda i,j : -2 if i == j and i not in {0,N} else \
      (1 if i in {j+1, j-1} and i not in {0,N} else 0)
      return np.fromfunction(np.vectorize(fill), (N+1,N+1))
```

6. 2.5 pts

$O(N^2)$, la fonction permet de remplir une matrice d'ordre (N+1)

7. 10 pts

Version 1 :

```
def SolutionNumerique(L,T,N,M,alpha):
    vx=np.linspace(0,L,N+1)
    dx=vx[1]-vx[0]
    vt=np.linspace(0,T,M+1)
    u0=np.zeros(N+1)
    for i in range(1,N+1): #ou bien u0=np.array([Tinitial(i) for i in vx])
      u0[i]=Tinitial(vx[i])
    A=GenererA(N)
    k= lambda u,t: alpha/(dx*dx)*np.dot(A,u)
    U=np.transpose(spi.odeint(k, u0, vt))
    return U,vx,vt
```

Version 2 :

```
def SolutionNumerique(L,T,N,M,alpha):
     vx, dx = np.linspace(0,L,N+1, retstep = True)
     vt= np.linspace(0, T, M+1)
     u0 = np.vectorize(Tinitial)(vx)
     A = GenererA(N)
     k = lambda u, t : (alpha/dx**2)* (A.dot(u))
     U = spi.odeint(k, u0, vt).T # ou bien spi.odeint(k, u0, vt).transpose()
     return (U, vx, vt)
```

Partie 3

8. 2.5 pts + 2.5 pts

```
class EqChaleur:
8.1 def __init__(self,L,T,N,M,alpha):
          self.L=L
          self.T=T
          self.N=N
          self.M=M
          self.alpha=alpha

8.2 def SolveEq(self):
          return SolutionNumerique(self.L,self.T,self.N,self.M,self.alpha)
```

9.

```
class InterpolationBilinieaire:
```

9.1 5 pts

```
def __init__(self,U,vx,vt):
   self.U = U.copy() # self.U=U
   self.bornes_max = (vx.max(), vt.max())
   self.pas_v = (vx[1]-vx[0], vt[1]-vt[0])
```

9.2 2.5 pts

Version 1:

```
def __str__(self) :
    motif = "F : [0, {}] x [0, {}] --> [{}, {}]"
    bxmax, btmax  = self.bornes_max
    return motif.format(bxmax, btmax, self.U.min(), self.U.max())
```

Version 2:

```
def __str__(self):
   xmax,tmax= self.bornes_max
```

```
    Umin,Umax=self.U.min(),self.U.max()
    ch= 'F:[0,'+str(xmax)+']*[0,'+ str(tmax)+'] -->\
                           ['+str(Umin)+','+str(Umax)+']'
   return ch
```

9.3 2.5 pts

Version 1:

```
def __contains__(self,tup):
  if 0< tup[0]< self. bornes_max [0] and 0< tup[1]< self. bornes_max [1]:
             return True
  else:
            return False
```

Version 2:

```
def __contains__(self, tup):
   x, t = tup
   xmax, tmax = self.bornes_max
   return 0 < x < xmax and 0 < t < tmax
```

9.4 2.5 pts

```
def get(self,tup):
      return (tup[0]/self.pas_v[0],tup[1]/self.pas_v[1])
```

9.5 1.25 pts

Version 1:

```
def getlow(self,tup):
      t1=self.get(tup)
      t2= math.floor(t1[0]),math.floor(t1[1])
      return t2
```

Version 2:

```
def get_low(self, tup):
      return tuple(math.floor(x) for x in self.get(tup))
```

9.6 1.25 pts

Version 1:

```
def getup(self,tup):
      t1=self.get(tup)
      t2= math.ceil(t1[0]),math.ceil(t1[1])
      return t2
```

Version 2:

```
def get_up(self, tup):
      return tuple(math.ceil(x) for x in self.get(tup))
```

9.7 10 pts

```
from np.linalg import solve
```

Version 1:

```
def interpolate(self,tup):
     if tup in self:
        xlow,tlow=self.getlow(tup)
        x_up,t_up=self.getup(tup)
        v=np.array([1,tup[0],tup[1],tup[0]*tup[1]])
        xl=xlow*self.pas_v[0]
        xu=x_up*self.pas_v[0]
        tl=tlow*self.pas_v[1]
        tu=t_up*self.pas_v[1]
        c1=np.ones(4,'int')
        c2=np.array([xl,xl,xu,xu])
        c3=np.array([tl,tu,tl,tu])
```

```
              B=np.array([c1,c2,c3,c2*c3])
              B=np.transpose(B)
              b=np.array([self.U[xlow,tlow],self.U[x_up,tlow],\
                                   self.U[xlow,t_up],self.U[x_up,t_up]])
              y=solve(B,b)
              return np.dot(y,v)
            else:
              return 'Erreur'
```

Version 2:

```
      def interpolate(self, tup):
        assert tup in self
        x, t  = tup
        dx, dt = self.pas_v
        x_low, t_low = self.get_low(tup)
        x_up, t_up = self.get_up(tup)
        v = np.array([1, x, t, x * t ])
        x_l, x_u, t_l, t_u  = np.array([x_low,x_up,t_low ,t_up]) * [dx,dx,dt,dt]
        B = np.ones((4,4))
        B[0,1] = B[1,1] = x_l
        B[2,1] = B[3,1] = x_u
        B[0,2] = B[2,2] = t_l
        B[1, 2] = B[3,2] = t_u
        B[:,-1] = B[:,-2] * B[:,-3]
        b = np.array([self.U[i,j] for i in (x_low, x_up) for j in (t_low,
t_up)])
        y = solve(B,b)
        return y.dot(v)
```

10. 7.5 pts

```
   chal=EqChaleur(L,T,N,M,alpha)
   U,vx,vt=chal.SolveEq()
   bil=InterpolationBilinieaire(U,vx,vt)
   while 1:
        try:
             x=float(input('lire x'))
             t=float(input('lire t'))
             if x not in vx and t not in vt:
                        break
        except:
             print('saisir des réels')
   print(bil.interpolate((x,t)))
```

PROBLEME 2

Partie 1

1. **2.5 pts**

$\pi_{nom}(\sigma_{idEtab\ =\ 'Libre'\ et\ section\ =\ 'PC'}(Candidat))$

2. **2.5 pts**

$\pi_{idC}(Candidat)$ - $\pi_{idC}(Evaluation)$

Partie 2

3. **2.5 pts**

update Epreuve set duree = 2 where nomEpr='Informatique' and section= 'T';

4. **2.5 pts**

select DISTINCT E.nom from Etablissement E join candidat C on E.idEtab = C.idEtab
where C.section ='BG' and E.idEtab <> 'Libre'

5. **3.75 pts**
   select section, sum(coeff) s from Epreuve group by section order by s desc;

6. **3.75 pts**
   SELECT IdC FROM Evaluation WHERE note >= 15 GROUP BY IdC Having count(*) >= 3;
   ou
   select idC, count(idEpr) s from evaluation where note >=15 group by idC having s>=3;

Partie 3 :

7. **2.5 pts**

```
def Notes(cur,id):
     cur.execute('select note from Evaluation where idEpr=?’, [id])
     L=cur.fetchall()
     return [i[0] for i in L]
```

8. **2.5 pts**

```
def ecart_type(cur,id):
     L=Notes(cur,id)
     m= sum(L)/len(L)
     s=0
     for i in L:
          s+=(i-m)**2
     return math.sqrt(s/len(L))
```

9. **2.5 pts**

```
def Epreuves(cur,s):
     cur.execute('select idEpr from Epreuve where section=?', (s,))
     L=cur.fetchall()
     return {i[0] for i in L}
```

10. **2.5 pts**

```
def ecartypeEpreuves(cur,s):
     req = """
     SELECT nomEpr FROM Epreuve WHERE IdEpr = ?
     """
     L=Epreuves(cur,s)
     d_ecart={}
     for i in L:
          cur.execute(req, [i])
          nom = cur.fetchone()[0]
          d_ecart[nom]=ecart_type(i)
     return d_ecart
```

11. 5 pts

```
def discriminante(cur,s):
     d=ecartypeEpreuves(cur,s)
     m=-1
     for i in d :
          if d[i]>m:
               m=d[i]
               ep=i
     return ep
```

Version 1

```
def discriminante(cur, s):
    d = EcartTypeEpreuves(cur, s)
    return max(d, key = lambda nomEpr : d[nomEpr])
```

version 2

```
def discriminante(cur, s):
    d = EcartTypeEpreuves(cur, s)
    ref = -1
    res = None
    for nomEpr in d:
        if d[nomEpr] > ref:
            ref = d[nomEpr]
            res = nomEpr
    return res
```

Version 3

```
def discriminante(cur, s):
    d = EcartTypeEpreuves(cur, s)
    ref = -1
    res = None
    for nomEpr, stdEpr in d.items():
        if stdEpr > ref:
            ref = stdEpr
            res = nomEpr
    return res
```