Problem 6.1 A is symmetric so we know that by the spectral theorem there exists an orthogonal matrix P such that

$$A = P\,\mathrm{Diag}(\lambda_1 \dots \lambda_n) P^T$$

where $\lambda_1, \dots \lambda_n$ are the eigenvalues of A.

A is orthogonal $\Leftrightarrow AA^T = Id_n \Leftrightarrow P\,\mathrm{Diag}(\lambda_1 \dots \lambda_n) P^T P\,\mathrm{Diag}(\lambda_1 \dots \lambda_n) P^T = Id$

$\Leftrightarrow \mathrm{Diag}(\lambda_1^2, \dots, \lambda_n^2) = Id_n$

$\Leftrightarrow |\lambda_i| = 1$ for all i.

Problem 6.2

a) Let $x \in \mathbb{R}^n$. $x^T AA^T x = (A^T x)^T A^T x = \|A^T x\|^2 \geq 0$.
Hence AA^T is positive semidefinite.

b) Assume that M is positive semidefinite. Let λ be an eigenvalue of M and $x \in \mathbb{R}^n$ be an associated eigenvector: $Mx = \lambda x$.
Compute $x^T Mx = \lambda x^T x = \lambda \|x\|^2$. $x \neq 0$ so we get

$$\lambda = \frac{x^T Mx}{\|x\|^2} \geq 0.$$

Conversely, assume that all the eigenvalues of M are non-negative. By the spectral theorem, there exists a orthonormal basis $(v_1, \dots v_n)$ of $\mathbb{R}^n$ such that all the v_i are eigenvectors of M: $Mv_i = \lambda_i v_i$, for some $\lambda_i \in \mathbb{R}$.

By assumption we get $\lambda_i \geq 0$ for all i. Let $x \in \mathbb{R}^n$ and let $(\alpha_1 \dots \alpha_n)$ be the coordinates of x in the basis $(v_1 \dots v_n)$

$x = \alpha_1 v_1 + \dots + \alpha_n v_n$.

Moltiplying by M gives: $Mx = \alpha_1 M v_1 + \dots + \alpha_n M v_n$
$= \alpha_1 \lambda_1 v_1 + \dots + \alpha_n \lambda_n v_n$

Hence $(\alpha_1\lambda_1, \dots \alpha_n\lambda_n)$ are the coordinates of Mx in the orthonormal basis $(v_1 \dots v_n)$:

$$x^T M x = \langle x, Mx \rangle = \sum_{i=1}^{n} \alpha_i \times \alpha_i \lambda_i$$

$$= \sum_{i=1}^{n} \alpha_i^2 \lambda_i \geqslant 0$$

M is positive semidefinite.

c) Let M be a symmetric positive semidefinite matrix.
By the spectral theorem we know that

$$M = P \begin{pmatrix} \lambda_1 & & (0) \\ & \ddots & \\ (0) & & \lambda_n \end{pmatrix} P^T$$ for some orthogonal

matrix P, where $\lambda_1 \geqslant \lambda_2 \geqslant \dots \geqslant \lambda_n$ are the eigenvalues of M. (We can assume that $\lambda_1 \dots \lambda_n$ are ordered in that way, otherwise it suffices to permute the columns of P).

From (b) we know that $\lambda_1 \dots \lambda_n \geqslant 0$. Since M is diagonalizable, we know also that

$$r = \operatorname{rank}(M) = \#\{i \mid \lambda_i \neq 0\}$$

We get that $\begin{cases} \lambda_1 \geqslant \lambda_2 \dots \geqslant \lambda_r > 0 \\ \lambda_{r+1} = \dots = \lambda_n = 0. \end{cases}$

Hence $M = P \begin{pmatrix} \lambda_1 & & & & (0) \\ & \ddots & & & \\ & & \lambda_r & & \\ & & & 0 & \\ (0) & & & & \ddots & 0 \end{pmatrix} P^T$.

Define $B = \begin{pmatrix} \sqrt{\lambda_1} & & (0) \\ (0) & \ddots & \sqrt{\lambda_r} \\ & (0) & \end{pmatrix} \in \mathbb{R}^{n \times r}$ we get

$M = PBB^T P^T = PB(PB)^T = AA^T$ where $A = PB \in \mathbb{R}^{n \times r}$.

<u>Problem 6.4</u>. For all $x \in \mathbb{R}^n$, $x^T A x = (x^T A x)^T = x^T A^T x$.

Hence $x^T A x = x^T \underbrace{\left(\frac{A + A^T}{2} \right)}_{M} x = x^T M x$. $(*)$

M is symmetric.

Since $x^T A x \geq 0$ for all x, we get that M is positive semi-definite.

By probleme 6.2, there exists $B \in \mathbb{R}^{n \times \mathrm{rank}(M)}$ such that $M = BB^T$

Let $x \in \mathrm{Ker}(A)$. From $(*)$ we get $x^T M x = 0$ so $x^T B B^T x = 0$ hence $B^T x = 0$, $BB^T x = 0$ so $Mx = 0$.

So

$$0 = Mx = \frac{1}{2} (\underbrace{Ax}_{=0} + A^T x)$$, which gives that $A^T x = 0 : x \in \mathrm{Ker}(A^T)$

We get that $\mathrm{Ker}(A) \subset \mathrm{Ker}(A^T)$. The inclusion $\mathrm{Ker}(A^T) \subset \mathrm{Ker}(A)$ follows by applying the result to A^T which verifies $x^T A x \geq 0$ for all $x \in \mathbb{R}^n$.

In [25]:

```
%matplotlib inline
import matplotlib.pyplot as plot
import csv
import numpy as np
plot.rc('font',family='serif')
plot.rc('xtick',labelsize=14)
```

In [26]:

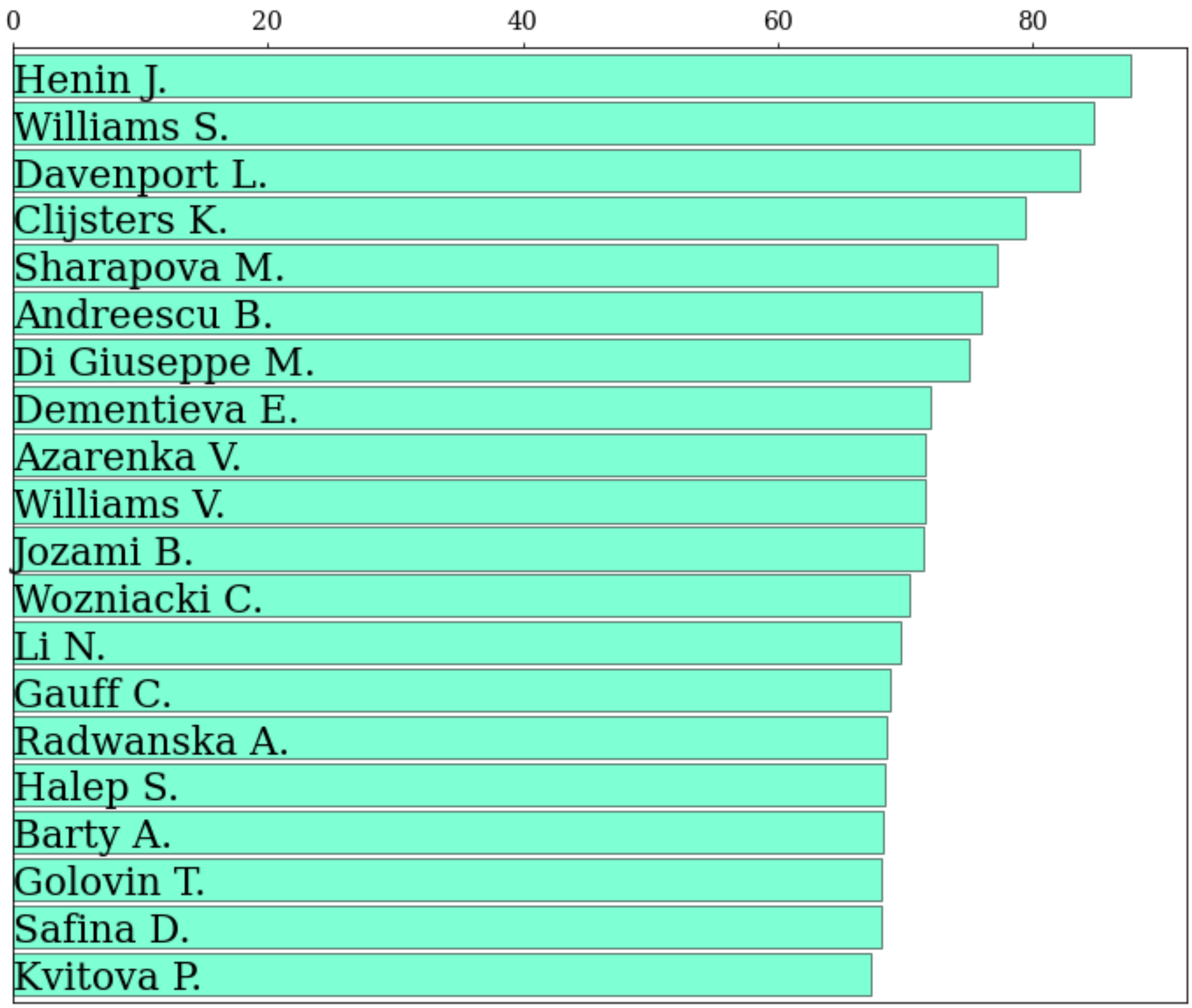
```
# The database contains the results of all tennis games
# in the pro men (ATP, from 2000 to end 2019) and women (WTA, from 2007 to en

# This codes reads the data
# Select the category: 'wta' for women, 'atp' for men
tour = 'wta'

# a setting to read the CSV files
if tour == 'atp':
    i_loser = 30
    i_winner = -2
else:
    i_loser = 21
    i_winner = -2

N=0                     # Total number of players (will be incremented when read
player_ID = dict()      # Given a 'name', player_ID[name] gives the ID of the pl
player_name=[]          # Given an 'id', player_name[id] gives the name of the p

# This reads the CSV file to construct N, player_ID and player_name
with open(tour+'.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for row in reader:
        loser = row[i_loser].rstrip().replace(',','')
        winner = row[i_winner].rstrip().replace(',','')

        for player in [winner,loser]:
            if not player in player_ID:
                player_ID[player]=N
                player_name.append(player)
                N +=1

# Matrix of the game records: R[i,j] will contain the number of time i beat j
R=np.zeros(shape=(N,N))

# This constructs R
with open(tour+'.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)

    for row in reader:
        # each row corresponds to a game
        loser = player_ID[row[i_loser].rstrip().replace(',','')]   # ID of th
        winner = player_ID[row[i_winner].rstrip().replace(',','')] # ID of th
        R[winner,loser] += 1  # count +1 victory for the winner
```

In [27]:
```
wins = np.sum(R,axis=1)     # total number of victories
losses = np.sum(R,axis=0)  # total number of losses
N_games = wins + losses    # total number of games
```

In [28]:
```
# naive ranking: rank players by percentage of victories
ratio = wins/N_games
naive_ranking = ratio.argsort()[::-1]
naive_scores = np.sort(ratio)[::-1]
```

In [29]:
```
# Function that plots rankings
def plot_ranking(ranking,scores,n):
    y=-np.array(range(n))
    plot.figure(figsize=(12,n/2),frameon=False)
    plot.barh(y,100*scores[:n],color='aquamarine', height=0.9, edgecolor = 'b
    for i in range(n):
        plot.text(0.0922,y[i]-0.35,player_name[ranking[i]],fontsize=22)
    t=plot.yticks([],[])
    l=plot.ylim(-n+ 0.4,0.6)
    ax = plot.gca()
    ax.xaxis.tick_top()
    #plot.savefig("ranking.pdf",bbox_inches='tight',transparent=True)
```

In [30]:
```
# Plot the 'naive' (ie in terms of percentage of victories) ranking of the to
plot_ranking(naive_ranking,naive_scores,20)
```



(a) Compute the transition matrix P as in the notes, then construct the matrix

$$M = \alpha P + \frac{1-\alpha}{N} J$$

where J is the all-one matrix, and $\alpha = 0.99$.

In [7]:
```
D = np.diag(1/N_games)
P = (R + np.diag(wins))@ D

alpha = 0.99
M = alpha*P + (1-alpha)*np.ones(shape=(N,N))/N
```

(b) Compute the stationary distribution of the Markov chain of transition matrix M.

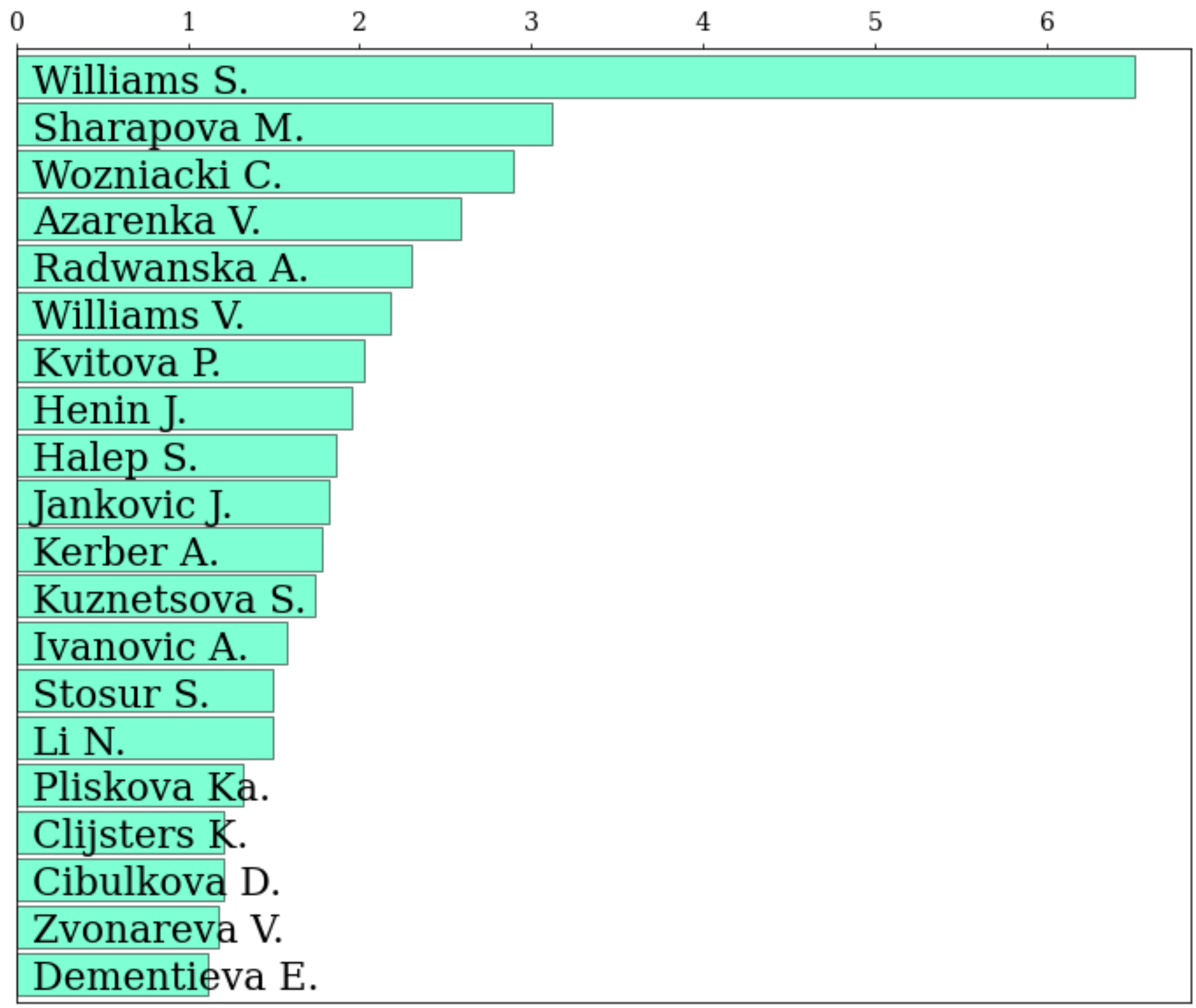
In [8]:
```
x = np.ones(N)/N
# Perron Frobenius Theorem guarantees that this converges:
while np.sum(np.square(x-P@x)) > 10**-20:
    x = P @ x

# (Optional) Since there can be rounding effects, I check that the sum of coe
# x is equal to 1:
print(np.sum(x))
```

```
0.9999999999999993
```

(c) Use the stationary distribution to rank the players, and plot the ranking of the best 20 players.

In [9]:
```
ranking = x.argsort()[::-1]
scores = np.sort(x)[::-1]
```

In [10]:
```
plot_ranking(ranking,scores,20)
```



(d) Open-ended question. For this question, no particular answer is awaited. Investigate the data (and maybe the wikipedia pages of the players, even though you do not need to know their careers by heart!), do some plots, other rankings, to find possible explanations to the following observations (for the women rankings):

- How would you explain that Henin, who is N1 in term of percentage of victories is way behind in page-rankings?
- Recompute the ranking, but now with $\alpha = 0.9$. How do you explain that Wozniacki is now ranked before Sharapova?

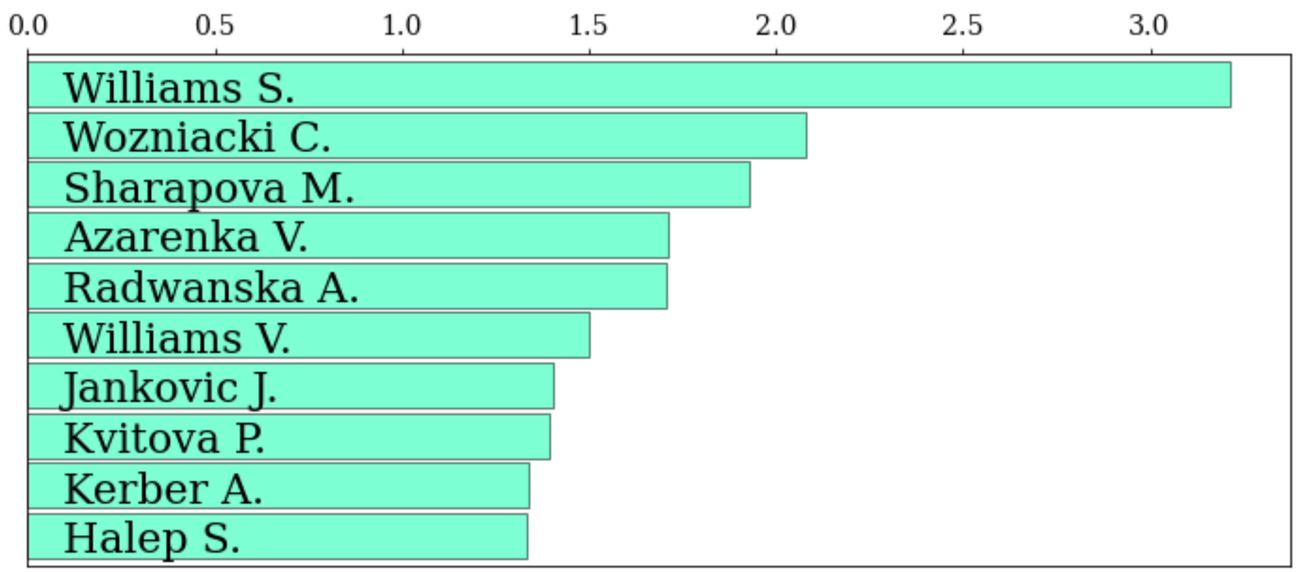
Henin dominated tennis from 2000 to 2009: since we have only access to the game after 2007, only a small fraction of her games are in the dataset. There is therefore few edges pointing to her, compared to the other players. But still, she is a good player so she has a great percentage of victories.

In [12]:
```
# One can verify this:

id_Henin = player_ID['Henin J.']
print(f'Henin has a total of {int(N_games[id_Henin])} games in the dataset')
print(f'While the top 10 has')
for j in range(10):
    i = ranking[j]
    print(f'{j+1}. {player_name[i]}: {int(N_games[i])} games.')
```

```
Henin has a total of 130 games in the dataset
While the top 10 has
1. Williams S.: 573 games.
2. Sharapova M.: 523 games.
3. Wozniacki C.: 833 games.
4. Azarenka V.: 599 games.
5. Radwanska A.: 731 games.
6. Williams V.: 523 games.
7. Kvitova P.: 595 games.
8. Henin J.: 130 games.
9. Halep S.: 531 games.
10. Jankovic J.: 672 games.
```

In [13]:
```
D = np.diag(1/N_games)
P = (R + np.diag(wins))@ D

alpha = 0.9 # New value for alpha
P = alpha*P + (1-alpha)*np.ones(shape=(N,N))/N

x = np.ones(N)/N
# Perron Frobenius Theorem guarantees that this converges:
while np.sum(np.square(x-P@x)) > 10**-20:
    x = P @ x

# (Optional) Since there can be rounding effects, I check that the sum of coe
# x is equal to 1:
print(np.sum(x))

ranking = x.argsort()[::-1]
scores = np.sort(x)[::-1]

plot_ranking(ranking,scores,10)
```

```
1.0000000000000322
```



Wozniacki is indeed now ranked before Sharapova. We see (from what we printed above) that Wozniacki played a total of 833 games (with 586 victories), while Sharapova has only 518 games in the databaset (with 400 victories).

Increasing α increases the probability of doing a 'jump' to an uniformly chosen player. This uniformly chosen player is unlikely to be Wozniacki or Sharapova. However, since Wozniacki has more victories that Sharapova, it is more likely to then jump from this player to Wozniacki than to Sharapova.

In [14]:
```
for j in range(5):
    i = ranking[j]
    print(f'{j+1}. {player_name[i]}: {int(wins[i])} victories.')
```

```
1. Williams S.: 486 victories.
2. Wozniacki C.: 586 victories.
3. Sharapova M.: 404 victories.
4. Azarenka V.: 429 victories.
5. Radwanska A.: 501 victories.
```

(e) Open-ended question. In fact, both CSV files contain the score (the number of sets won by each of the players) of each game. Propose a method based on PageRank, but with another transition matrix P, that takes the scores into account, in order to obtain more 'accurate' rankings and implement it. There is no particular method expected. Your are only suppose to propose something 'coherent' (for instance winning games by a large margin should improve rankings...)

In [19]:

```
# This code opens the game database
# it loops over all the games
# for each game it extracts the 'id' of the winner/loser
# and the number of sets won by each player

R=np.zeros(shape=(N,N))
with open(tour+'.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)

    for row in reader:
        # each row corresponds to a game
        loser = player_ID[row[i_loser].rstrip().replace(',','')]   # ID of th
        winner = player_ID[row[i_winner].rstrip().replace(',','')] # ID of th

        # check if the number of sets for each player is available
        if row[i_loser+1] != '' and row[i_winner+1] != '':
            loser_sets = int(float(row[i_loser+1]))
            winner_sets = int(float(row[i_winner+1]))
            if winner_sets == 0:
                # For some games (where one of the players retired because of
                # The number of sets is 0. In that case we say that the winne
                winner_sets = 2
                loser_sets = 0
        else:
            # if the number of sets are not available, we say that the winner
            loser_sets = 0
            winner_sets = 2

        # Do something with loser_sets, winner_sets

        # Proposed method: a victory with a difference of 'x' sets
        # has a weight proportional to 'x'

        R[winner,loser] += winner_sets - loser_sets
```

In [20]:

```
# Implement your method
```

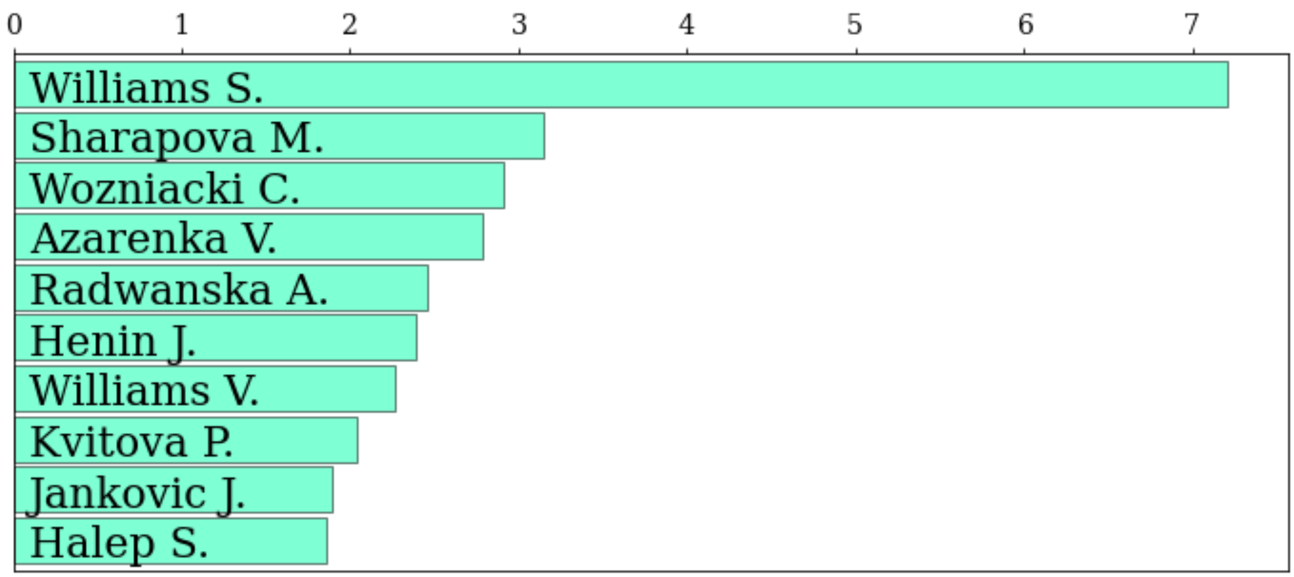
In [23]:

```
# Then, we compute the total number of 'victory/loss weight'
# for each player (note that since a big victory count twice, this does not
# correspond to the number of victories/losses anymore).

wins = np.sum(R,axis=1)     # total number of victories
losses = np.sum(R,axis=0)   # total number of losses
N_games = wins + losses     # total number of games
```

In [24]:

```
D = np.diag(1/N_games)
P = (R + np.diag(wins))@ D

alpha = 0.99 # New value for alpha
P = alpha*P + (1-alpha)*np.ones(shape=(N,N))/N

x = np.ones(N)/N
# Perron Frobenius Theorem guarantees that this converges:
while np.sum(np.square(x-P@x)) > 10**-20:
    x = P @ x

# (Optional) Since there can be rounding effects, I check that the sum of coe
# x is equal to 1:
print(np.sum(x))

ranking = x.argsort()[::-1]
scores = np.sort(x)[::-1]

plot_ranking(ranking,scores,10)
```

```
1.0000000000000424
```



In [ ]: