Exemplos de processamento de dados em Hidrologia

Nelson Luís Dias[1]

[1]Departamento de Engenharia Ambiental, UFPR.

30 de julho de 2023

1 Arquivos de dados: repositórios úteis e formatos

Atualmente, muitos dos "métodos" desenvolvidos historicamente para hidrologia estão se tornando, ou se tornaram, obsoletos, com o advento de amplos recursos computacionais, com a expansão das redes de monitoramento, com o surgimento de sistemas de informação geográfica, e com o advento do grandes bases de dados de reanálise e de sensoriamento remoto.

Algumas bases globais são extremamente úteis, e devem ser citadas explicitamente:

SRTM (Shuttle Radar Topograpphy Mission) Veja `https://www2.jpl.nasa.gov/srtm/`. Dados de topografia com resolução de 30 m, para todo o planeta.

MODIS Um grande número de produtos de sensoriamento remoto (satélite) (Ver tabelas a seguir)

CSFV2 (Dados de reanálise). Ver `https://rda.ucar.edu/`. Dados em formato NETCDF.

ERA5 (Dados de reanálise). Ver `https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels?tab=overview`.

Endereços de obtenção de produtos MODIS úteis para estimativas espacializadas de evaporação

Variável	Endereço
r_0	`https://modis.gsfc.nasa.gov/data/dataprod/mod09.php`
T_0, ϵ_0	`https://modis.gsfc.nasa.gov/data/dataprod/mod11.php`
NDVI	`https://modis.gsfc.nasa.gov/data/dataprod/mod13.php`
E	`https://modis.gsfc.nasa.gov/data/dataprod/mod16.php`

Frequência e resolução de produtos disponíveis

Variável	Frequência	Resolução espacial
r_0	8 dias	250 m
r_0	8 dias	500 m
r_0	1 dia	1 km
r_0	1 dia	250 m
T_{0r}, ϵ_0	8 dias	1 km
T_{0r}, ϵ_0	1 dia	1 km
NDVI	16 dias	250 m
NDVI	16 dias	500 m
NDVI	16 dias	1 km
E	8 dias	500 m

Nós precisamos, portanto, de capacidade para acessar as bases de dados, recuperar a informação relevante, entender e traduzir formatos, e realizar os processamentos necessários.

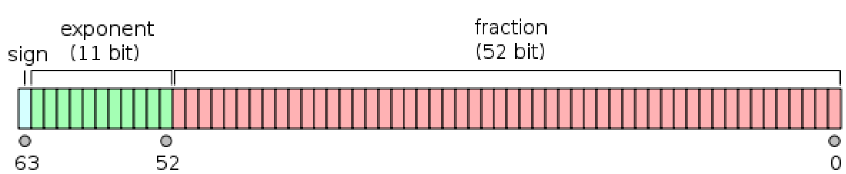
Nesta aula, nós vamos aprender alguns conceitos básicos que podem ser úteis para nós, embora a maior parte do processamento no curso vá ser feito com ferramentas e com formatos relativamente simples.

A escolha de uma linguagem de programação

No fundo, qualquer uma serve. Use a que você dominar melhor. Minha preferência pessoal, principalmente para um curso como o nosso, definitivamente é Python. Todos os meus exemplos neste curso serão em Python. Minha sugestão de instalação é miniconda: `https://docs.conda.io/en/latest/miniconda.html`.

Formatos binários, e formatos texto

Num formato binário, os dados são gravados como estavam na memória. Em geral isso significa que cada 8 bytes = 64 bits (tipicamente) codificam um número de ponto flutuante. Em computadores com processadores Intel, isso em geral significa o esquema abaixo:



A faixa de números que podem ser representados em 64 bits com essa organização é a seguinte:

$$\pm 2^{51} \times 2^{\pm 2^{10}}$$

A *mantissa* é um número inteiro de 52 bits. O *expoente* é outro número inteiro de 10 bits (mais um bit de sinal). O maior número representável em módulo dessa forma é

$$\pm 2^{51} \times 2^{1023} = \pm 2.024022533073106 \times 10^{323},$$

mas na prática o menor e o maior valor representáveis são

$$\pm 1.79769 \times 10^{308}.$$

Além disso (no sentido de $\pm\infty$) ocorre *overflow*.

Os números mais próximos de 0 representáveis são

$$\pm 2.22507 \times 10^{-308}.$$

Além disso (no sentido de 0) ocorre *underflow*.

O comportamento do programa quando ocorre *overflow*, *underflow* ou quando ocorrem operações ilícitas (por exemplo, $\sqrt{-1.0}$ para números "reais") é indefinido, e depende da linguagem e muitas vezes das opções com que o programa é rodado.

Um ponto importante a ser levado em consideração é que o fato de a representação interna ser toda na base 2 leva a erros de arredondamento. Considere o programa em Python

```
#!/home/nldias/miniconda3/bin/python3
# -*- coding: iso-8859-1 -*-
x = 0.1
print('%40.38f' % x)
```

sua saída é

```
0.10000000000000000555111512312578270212
```

Devido aos erros de arredondamento a se passar da base 10 para a base 2, o programa

```
#!/home/nldias/miniconda3/bin/python3
# -*- coding: iso-8859-1 -*-
x = 0.0
while True :
   x = x + 0.1
   print(x)
   if (x == 1.0) :
      break
   pass
pass
```

não para nunca!

Dito isso, vamos fazer alguns exemplos de arquivos binários. Nosso primeiro exemplo vai escrever 10 números de ponto flutuante em formato binário no disco, e depois nós vamos ver o seu "conteúdo". O programa é

```
#!/usr/bin/python3
fou = open('bbin.dat','wb')      # abre um arquivo. wb == write/binary
from numpy import arange         # arange devolve um array com incrementos fixos
aa = arange(0.0,10.0,0.1,float) # cria o array
aa.tofile(fou)                   # escreve o array no arquivo fou
fou.close()                      # fecha o arquivo
print('%6.4f' % aa[0])
```

A saída agora, se visualizada, é

```
^@^@^@^@^@^@^@^@<9A><99><99><99><99><99><B9>?<9A><99><99><99><99><99>
<C9>?433333<D3>?<9A><99><99><99><99><99><D9>?^@^@^@^@^@^@<E0>?433333
<E3>?gfffff<E6>?<9A><99><99><99><99><99><E9>?<CD><CC><CC><CC><CC><CC>
<EC>?^@^@^@^@^@^@<F0>?<9A><99><99><99><99><99><F1>?433333<F3>?<CD><CC>
<CC><CC><CC><CC><F4>?gfffff<F6>?^@^@^@^@^@^@<F8>?<9A><99><99><99><99>
```

```
<99><F9>?433333<FB>?<CD><CC><CC><CC><CC><CC><FC>?gfffff<FE>?^@^@^@^@^@
^@^@@<CD><CC><CC><CC><CC><CC>^@@<9A><99><99><99><99><99>^A@gfffff^B@43
3333^C@^@^@^@^@^@^@^D@<CD><CC><CC><CC><CC><CC>^D@<9A><99><99><99><99>
<99>^E@gfffff^F@433333^G@^@^@^@^@^@^@^H@<CD><CC><CC><CC><CC><CC>^H@<9A>
<99><99><99><99><99>@gfffff@433333^K@^@^@^@^@^@^@^L@<CD><CC><CC><CC>
<CC><CC>^L@<9A><99><99><99><99><99>^M@gfffff^N@433333^O@^@^@^@^@^@^@^P@
gfffff^P@<CD><CC><CC><CC><CC><CC>^P@333333^Q@<9A><99><99><99><99><99>^Q
@^@^@^@^@^@^@^R@gfffff^R@<CD><CC><CC><CC><CC><CC>^R@433333^S@<9A><99><99>
<99><99><99>^S@^@^@^@^@^@^@^T@gfffff^T@<CD><CC><CC><CC><CC><CC>^T@433333^
U@<9A><99><99><99><99><99>^U@^@^@^@^@^@^@^V@gfffff^V@<CD><CC><CC><CC><CC>
<CC>^V@433333^W@<9A><99><99><99><99><99>^W@^@^@^@^@^@^@^X@gfffff^X@<CD>
<CC><CC><CC><CC><CC>^X@433333^Y@<9A><99><99><99><99><99>^Y@^@^@^@^@^@^@
^Z@gfffff^Z@<CD><CC><CC><CC><CC><CC>^Z@433333ESC@<9A><99><99><99><99><99>
ESC@^@^@^@^@^@^@^\@gfffff^\@<CD><CC><CC><CC><CC><CC>^\@433333^]@<9A><99>
<99><99><99><99>^]@^@^@^@^@^@^@^^@gfffff^^@<CD><CC><CC><CC><CC><CC>^^@
433333^_@<9A><99><99><99><99><99>^_@^@^@^@^@^@^@@333333@gfffff@<9A><99>
<99><99><99><99>@<CD><CC><CC><CC><CC><CC>@^@^@^@^@^@^@!@333333!@gfffff!
@<9A><99><99><99><99><99>!@<CD><CC><CC><CC><CC><CC>!@^@^@^@^@^@^@"@3333
33"@gfffff"@<9A><99><99><99><99><99>"@<CD><CC><CC><CC><CC><CC>"@^@^@^@^@
^@^@#@433333#@gfffff#@<9A><99><99><99><99><99>#@<CD><CC><CC><CC><CC><CC>#@
```

Note que o tamanho do arquivo `bbin.dat` é 800 bytes. Faz sentido, certo?

Por outro lado, nós podemos imprimir os 10 elementos do array `aa`, depois que ele foi criado, em um arquivo texto, com o seguinte programa

```
#!/usr/bin/python3
fou = open('ttxt.dat','wt')      # abre um arquivo. wt == write/text
from numpy import arange         # arange devolve um array com incrementos fixos
aa = arange(0.0,10.0,0.1,float) # cria o array
for i in range(0,10):           # loop sobre todos os elementos do array
   fou.write('%02d %4.1f\n' % (i,aa[i]))   # escreve cada elemento, pulando linha
pass
fou.close()                      # fecha o arquivo
```

cuja saída é

```
00  0.0
01  0.1
02  0.2
03  0.3
04  0.4
05  0.5
06  0.6
07  0.7
08  0.8
09  0.9
```

Finalmente, vamos ler o arquivo texto de volta:

```
#!/usr/bin/python3
fin = open('ttxt.dat','rt')      # abre um arquivo. wt == read/text
from numpy import zeros          # zeros devolve um array com zeros
aa = zeros(10,float)             # cria o array
for line in fin:                 # loop sobre as *linhas*
   field = line.split()          # separa os campos
   i = int(field[0])             # o primeiro campo
   aa[i] = float(field[1])       # o segundo campo
   print(field,i,aa[i])          # o que lemos?
pass
fin.close()                      # fecha o arquivo
```

A saída é

```
['00', '0.0'] 0 0.0
['01', '0.1'] 1 0.1
['02', '0.2'] 2 0.2
['03', '0.3'] 3 0.3
['04', '0.4'] 4 0.4
['05', '0.5'] 5 0.5
['06', '0.6'] 6 0.6
['07', '0.7'] 7 0.7
['08', '0.8'] 8 0.8
['09', '0.9'] 9 0.9
```

2 Chuva e vazão, gráficos

Tendo feito uma rápida "introdução" a Python (veja também os capítulos sobre Python em meu livro, em: https://nldias.github.io/pdf/matappa-2ed.pdf), vamos agora fazer um problema do livro-texto.

(Ex. 2.3.2 de Chow et al. (1988))

The precipitation and streamflow for the storm of May 12, 1980, on Shoal Creek at Northwest Park in Austin, Texas, are shown below. Calculate the time distribution of storage on the watershed assuming that the initial storage is 0. Compute the total depth of precipitation and the equivalent depth of streamflow which occurred during the 8-hour period. How much storage remained in the watershed at the end of the period? What percent of the precipitation appeared as streamflow during this period? What was the maximum storage? Plot the time distribution of incremental precipitation, streamflow, change in storage, and cumulative storage. The watershed area is 7.03 mi^2.

Time (h)	0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	
Incremental Precipitation (in)		0.18	0.42	0.21	0.16				
Instantaneous Streamflow (cfs)	25	27	38	109	310	655	949	1060	
Time (h)	4.0	4.5	5.0	5.5	6.0	6.5	7.0	7.5	8.0
Instantaneous Streamflow (cfs)	968	1030	826	655	466	321	227	175	160

Primeiro, nós preparamos um arquivo texto com os dados:

```
# Time (h) Precip (in)  Streamflow (cfs)
0.0        0.00         25.0
0.5        0.18         27.0
1.0        0.42         38.0
1.5        0.21         109.0
2.0        0.16         310.0
2.5        0.00         655.0
3.0        0.00         949.0
3.5        0.00         1060.0
4.0        0.00         968.0
4.5        0.00         1030.0
5.0        0.00         826.0
5.5        0.00         655.0
6.0        0.00         466.0
6.5        0.00         321.0
7.0        0.00         227.0
7.5        0.00         175.0
8.0        0.00         160.0
```

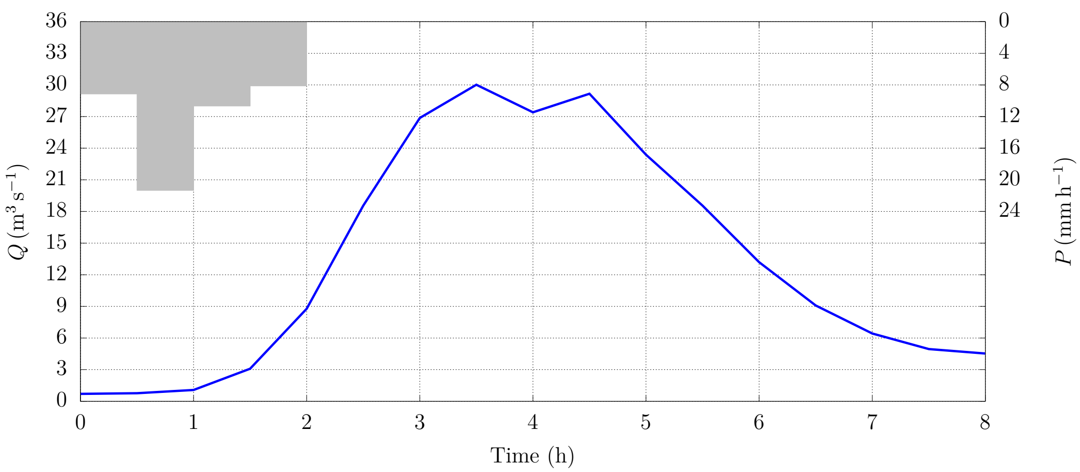
Em seguida nós plotamos os dados com Gnuplot

Os dados de entrada são plotados por

```
set encoding iso_8859_1
```

```
set terminal epslatex standalone color solid font 'lmr' 12 size 21cm, 9cm
set output 'shoaldata.tex'
set xrange [0:8]
set yrange [0:36]
set ytics 0,3
set y2range [48:0]
set xtics 0,1
set y2tics 0,4,24
set xlabel 'Time␣(h)'
set y2label '$P\,(\mathrm{mm\,h^{-1}})$'
set ylabel '$Q\,␣(\mathrm{m^3\,s^{-1}})$'
set boxwidth 1.0 relative
set grid
plot 'shoal.dat' using (column(1)-0.25):(column(2)*25.4*2) axes x1y2 \
      notitle with boxes fs solid lc rgb 'gray75',\
      'shoal.dat' using 1:(column(3)*0.3048**3) axes x1y1 \
      notitle with lines lt 1 lw 5 lc rgb 'blue'
```

e o gráfico resultante é este aqui:



O programa de processamento de dados está aqui:

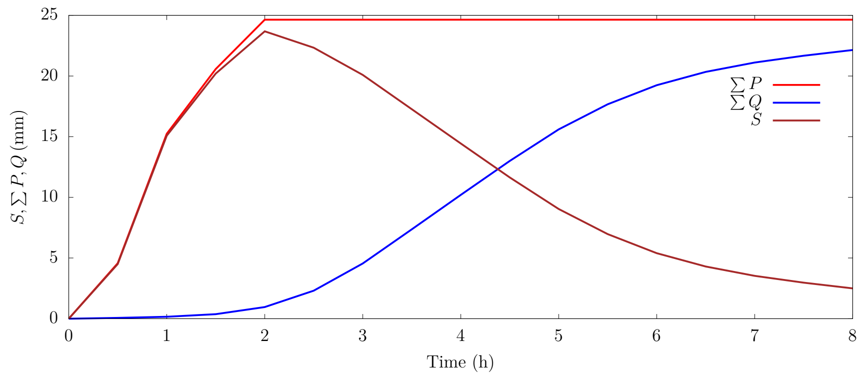
```
#!/home/nldias/miniconda3/bin/python3
# -*- coding: iso-8859-1 -*-
fin = open('shoal.dat','rt')       # abre um arquivo. wt == read/text
tt = []                            # lista vazia de tempos
pp = []                            # lista vazia de precipitações
qq = []                            # lista vazia de vazões
for line in fin:                   # loop sobre as *linhas*
   if line[0] == '#' :             # pula o cabeçalho
      continue
   pass
   field = line.split()            # separa os campos
   inst = float(field[0])          # o instante
   prec = float(field[1])*25.4     # precip em mm durante o intervalo
   vaz = float(field[2])*0.3048**3 # vazão em m3/s no instante
   tt.append(inst)                 # adiciona à lista de tempos
   pp.append(prec)                 # adiciona à lista de precips
   qq.append(vaz)                  # adiciona à lista de vazões
pass
fin.close()                        # fecha o arquivo de entrada
n = len(tt)                        # tamanho das listas
from numpy import array
tt = array(tt)                     # as listas se tornam arrays
pp = array(pp)
qq = array(qq)
# --------------------------------------------------------------------
# área da bacia, em m^2
```

```
# ----------------------------------------------------------------------
Area = 7.03 * (1609.34)**2
print('Area = %8.2f km2' % (Area/1.0e6))
qe = (qq/Area)*1000*3600.0         # vazão específica, mm/h
print(pp)
print(qe)
# ----------------------------------------------------------------------
# calculo o volume total precipitado
# ----------------------------------------------------------------------
Vp = pp.sum()
print('Vp = ', Vp, ' mm')
# ----------------------------------------------------------------------
# calculo o volume total escoado com a regra do trapézio
# ----------------------------------------------------------------------
deltah = 0.5                        # dados de 0.5 em 0.5 hora
Se = qe[0] + qe[n-1]
Si = 0.0
for k in range(1,n-1):
   Si += qe[k]
Vq = Se + 2*Si
Vq *= deltah
Vq /= 2.0
print('Vq = ', Vq, ' mm')
# ----------------------------------------------------------------------
# cálculo do volume armazenado na bacia em função do tempo
# ----------------------------------------------------------------------
fou = open('shoal.out','wt')
Vp = 0.0
Vq = 0.0
Vs = 0.0
fou.write('%4.2f %6.2f %6.2f %6.2f %6.2f\n' % (0.0,0.0,0.0,0.0,0.0))
for k in range(1,n):
   dp = pp[k]
   dq = (qe[k-1]+qe[k])*deltah/2.0
   Vp += dp
   Vq += dq
   deltas = dp - dq
   Vs += deltas
   fou.write('%4.2f %6.2f %6.2f %6.2f %6.2f\n' % (tt[k],Vp,Vq,Vs,deltas))
pass
fou.close()
```

O *script* de Gnuplot para os totais acumulados na bacia é

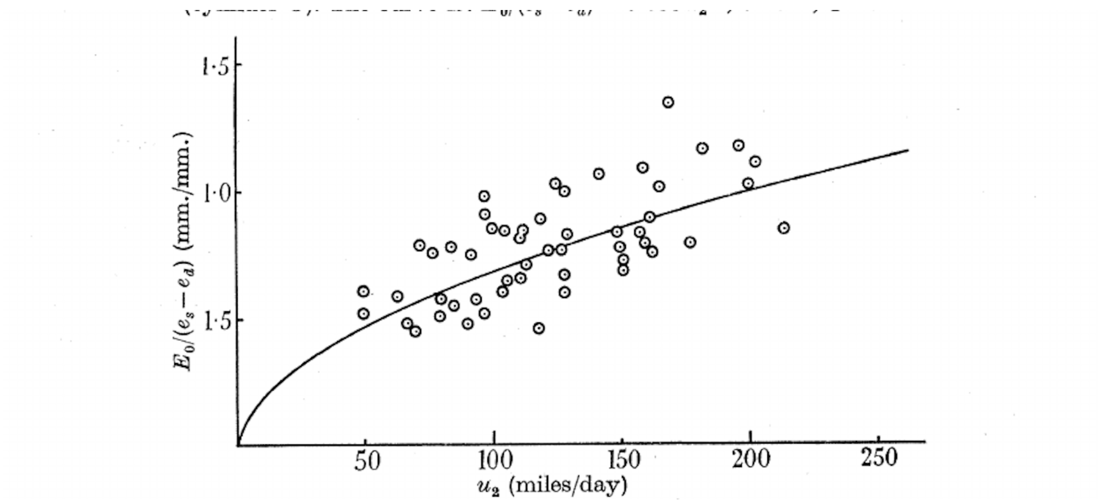
```
set encoding iso_8859_1
set terminal epslatex standalone color solid font 'lmr' 12 size 21cm, 9cm
set output 'shoalacum.tex'
set xrange [0:8]
set xtics 0,1
set xlabel 'Time (h)'
set ylabel '$S,\sum P,Q\,(\mathrm{mm})$'
set key at 6.75,20 left
plot 'shoal.out' using 1:2 title '$\sum P$' with lines lt 1 lw 5 lc rgb 'red',\
     'shoal.out' using 1:3 title '$\sum Q$' with lines lt 1 lw 5 lc rgb 'blue',\
     'shoal.out' using 1:4 title '$S$'      with lines lt 1 lw 5 lc rgb 'brown'
```

e o gráfico resultante é



3 Conversão de uma figura em uma tabela de dados

Desejamos converter a figura 3 de Penman (1948) em uma tabela. Reproduzimos a figura 3 aqui:



A tabela deve ser gravada em um arquivo **texto**. No artigo de Penman, as unidades do dados são as seguintes:

$\dot{h}_E$	(Taxa de evaporação) $\mathrm{mm_{H_2O}\,dia^{-1}}$
$(e_0^* - e_a)$	(Diferença de pressão de vapor d'água) $\mathrm{mm_{Hg}}$
u_2	(Velocidade do vento a 2 m de altura) $\mathrm{mi\,dia^{-1}}$

Algumas observações:

1. Quando o intervalo de tempo é um dia, a taxa de evaporação é numericamente igual à altura de água evaporada h_E, pois

$$h_E = \dot{h}_E \times \Delta t;$$

por isso, as unidades relatadas no eixo vertical estão em "mm/mm", ou seja: $\mathrm{mm_{H_2O}\,mm_{Hg}^{-1}}$.

2. Existe um erro na figura: os valores das marcas do eixo vertical são, de baixo para cima, 1.5, 1.0 e 1.5; obviamente, deveriam ser 0.5, 1.0 e 1.5.

No artigo de Penman, a figura é utilizada para calibrar a "Lei de Dalton" (mais corretamente, a equação de Stelling 1882):

$$\dot{h}_E = (a + bu_b)(e_0 - e_a), \tag{1}$$

onde $\dot{h}_E$ é a taxa de evaporação (altura de água evaporada por unidade de tempo), u_b é a velocidade do vento à altura z_b, e_0 é a pressão de vapor d'água na superfície da água, e e_a é a pressão de vapor d'água no ar à altura z_a. Em (1), a e b são constantes de calibração. Desejamos reescrever (1) de uma forma mais racional:

$$E = \rho(A + Bu_b)(q_0 - q_a) \tag{2}$$

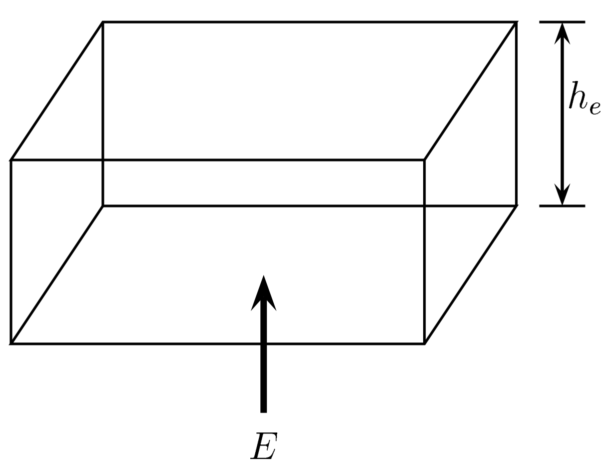
onde E é o fluxo de massa de vapor d'água, ρ é a densidade do ar, e q_0 e q_a são as umidades específicas de vapor d'água no ar correspondentes a e_0 e e_a.

4 Plano de ação

1. Definir as conversões de unidades (às vezes isso precisa ser refeito muitas vezes, porque **sempre** gera confusão).
2. Digitalizar os pontos da figura.
3. Gerar um arquivo de dados nas unidades originais da equação (1).
4. Converter para as unidades desejadas na equação (2).
5. Etc..

5 Conversões

Considere a figura a seguir:



$$
\begin{aligned}
E &= \frac{M}{A\Delta t};\\
M &= EA\Delta t = \rho_w A\dot{h}_E\Delta t;\\
E &= \rho_w \dot{h}_E\\
&= \frac{1000\,\mathrm{kg\,m^{-3}}10^{-3}\mathrm{m\,mm^{-1}}}{86400\,\mathrm{s\,dia^{-1}}} h_E\,\mathrm{mm\,dia^{-1}}\\
&= \frac{h_E}{86400}\,\mathrm{kg\,m^{-2}\,s^{-1}}
\end{aligned}
$$

$$
\begin{aligned}
1\mathrm{mm\,Hg} &= \rho_{\mathrm{Hg}} \times g \times 1\mathrm{mm}\\
&= 13534 \times 9.81 \times 10^{-3} = 132.76854\,\mathrm{Pa}
\end{aligned}
$$

Mas na verdade usamos o Google e um valor ligeiramente diferente!

$$
1\mathrm{mm\,Hg} = 132.322\,\mathrm{Pa}
$$

$$
1\ \mathrm{mi\,dia^{-1}} = \frac{1609\,\mathrm{m}}{24 \times 3600\,\mathrm{s}} = 0.018623\,\mathrm{m\,s^{-1}}
$$

Digitamos os pontos da figura com um programa adequado (em Linux, `g3data`), e produzimos o arquivo `Edpen.da0`

Listing 1: `Edpen.da0` (5 primeiras linhas)

```
49.3470340597  0.341587630276
49.4095468797  0.399662040067
70.018753846  0.295615311734
66.8110647691  0.31567215933
62.8507799451  0.3865675578
```

Note que as unidades de `Edpen.da0` correspondem à Equação (1). Ainda precisamos:

1. Obter um valor adequado para ρ.
2. Converter pressão de vapor em umidade específica.

Para calcular ρ, nós utilizamos uma atmosfera padrão e desconsideramos a umidade do ar (supomos ar seco):

$$
\begin{aligned}
h &= 128.02\,\mathrm{m}, \qquad \text{(altitude de Rothamstead)}\\
T_h &= 288.15 - 0.0065h,\\
P_h &= P_0\left(\frac{T_h}{288.15}\right)^{5.256} = 99796.51\,\mathrm{Pa},\\
P_h &= \rho R_d T_h,\\
\rho &= 1.21\,\mathrm{kg\,m^{-3}}.
\end{aligned}
$$

Para calcular a umidade específica q utilizamos

$$q = 0.622 \frac{e}{P_h}$$

Note que o eixo vertical da figura contém $\dot{h}_E/(e_0 - e_a)$. Com isso, podemos escrever um pequeno programa para gerar o mesmo conjunto de dados, porém nas unidades SI que desejamos:

Listing 2: `Edpen.awk` (conversão para o SI)

```
1  #!/usr/bin/gawk -f
2  # -------------------------------------------------------------------------------
3  # run me as
4  # ./Edpen.awk Edpen.da0 > Edpen.dat
5  # -------------------------------------------------------------------------------
6  BEGIN {
7     xtimes = 0.018623;
8     ytimes = 99796.51/(132.322*0.622*86400.0);
9     ytimes /= 1.21;  # density of air!
10    ytimes *= 1.5;   # because g3data screwed up
11 }
12 { printf("%7.5f␣%11.5e\n", $1*xtimes, $2*ytimes) }
```

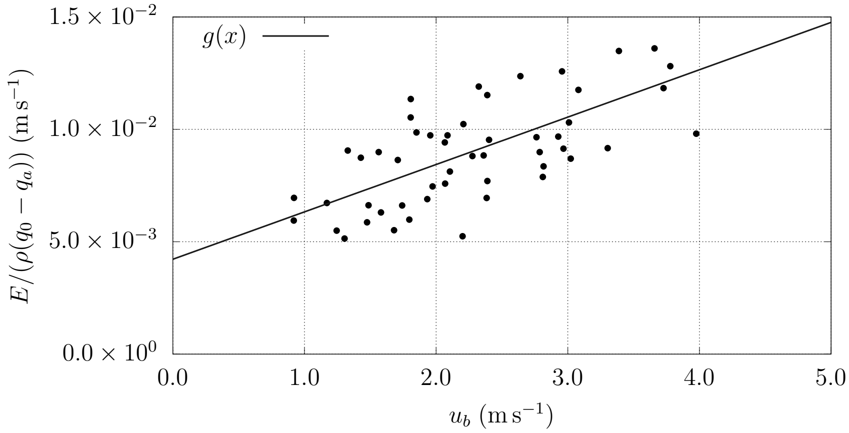
Finalmente: digitalizamos com uma escala de 0 a 1 em y, e não de 0 a 1.5; por isso, a linha 10 do programa!

Agora, a partir do arquivo de dados `Edpen.dat`, plotamos com

Listing 3: `Edpenf.plt`

```
load 'terminal.plt'
set terminal epslatex standalone color font 'lmr' 12 size 15cm, 7.5cm
set output 'Edpenf.tex'
set lmargin 12
g(x) = A + B*x
fit g(x) 'Edpen.dat' using 1:2 via A,B
set xrange [0:5]
set yrange [0:0.015]
set xtics 0,1
set ytics 0,0.005
set format y '$%3.1t␣\times␣10^{%T}$'
set format x '$%3.1f$'
set yzeroaxis
set grid
set key left
set xlabel '$u_b␣\;␣\mathrm{(m\,s^{-1})}$'
set ylabel '\vspace*{-1.25cm}$E/(\rho(q_0␣-␣q_a))\;\mathrm{(m\,s^{-1})}$'
plot 'Edpen.dat' using 1:2 notitle  with points pt 7 lc rgb 'black',\
     g(x) with lines lt 1 lw 3 lc rgb 'gray10'
```

E produzimos a figura



6 Conversão de unidades

Em Hidráulica, a equação de descarga para um vertedor é

$$Q = CLH^{3/2},$$

onde Q é a vazão, C é o coeficiente de descarga, L é a largura do vertedor e H é a carga sobre a soleira. No sistema britânico de unidades (Q em $\mathrm{ft^3\,s^{-1}}$, L e H em ft), $C \sim 2.8$. Obtenha C no SI.

SOLUÇÃO

$$\begin{aligned}
(\mathrm{m}/0.3048)^3\,\mathrm{s}^{-1} &= 2.8 \times (\mathrm{m}/0.3048)^{5/2}, \\
\mathrm{m}^3\,\mathrm{s}^{-1} &= 2.8 \times \frac{(0.3048)^3}{(0.3048)^{5/2}}\mathrm{m}^{5/2}; \;\Rightarrow \\
C &= 2.8 \times 0.3048^{1/2} = 1.546.
\end{aligned}$$

Uma outra abordagem é tentar um ataque *racional* e dimensionalmente consistente para o problema. Note que a equação

$$Q = CLH^{3/2}$$

é *dimensionalmente inconsistente*: uma evidência disso é que C muda de valor quando mudamos o sistema de unidades. É relativamente fácil corrigir isso, entretanto. Na equação acima a vazão cresce linearmente com a largura da soleira; portanto, basta considerar a *vazão por unidade de largura*, $q = Q/L$. Esta por sua vez depende claramente de H. Com um pequeno esforço, notamos que o escoamento é forçado apenas pela gravidade; incluímos portanto a aceleração da gravidade g na lista de variáveis intervenientes. Temos agora 3 variáveis (q, g e H) e 2 dimensões fundamentais: o comprimento L e o tempo T. A lista de dimensões das variáveis é

$$\begin{aligned}
[\![q]\!] &= \mathsf{L}^2\mathsf{T}^{-1}, \\
[\![g]\!] &= \mathsf{L}\mathsf{T}^{-2}, \\
[\![H]\!] &= \mathsf{L}.
\end{aligned}$$

A matriz dimensional é

	q	g	H
L	2	1	1
T	-1	-2	0

Existe apenas um parâmetro adimensional, que tem que ser constante; portanto,

$$\frac{q}{H\sqrt{gH}} = \alpha \qquad \Rightarrow \qquad Q = \alpha\sqrt{g}LH^{3/2}.$$

Uma segunda forma de responder à questão sobre o valor de C no SI, portanto, é reconhecer que $C = \alpha\sqrt{g}$ (em qualquer sistema de unidades!), onde agora α é uma constante adimensional e *universal*. No sistema britânico, $g = 32.2\,\mathrm{ft\,s^{-2}}$; logo,

$$\begin{aligned} \alpha\sqrt{32.2} &= 2.8; \\ \alpha &= \frac{2.8}{\sqrt{32.2}} = 0.493. \end{aligned}$$

Portanto, no SI nós revertemos o raciocínio:

$$C = \alpha\sqrt{g} = 0.493 \times \sqrt{9.81} = 1.545 \blacksquare$$

---

7 Propagação de cheias em um reservatório

Um reservatório de acumulação de cheias urbano tem uma área horizontal $A = 100000\,\mathrm{m}^2$ e paredes verticais. O reservatório está inicialmente vazio, e recebe uma cheia $I(t)$ mostrada em vermelho na figura 1. O reservatório possui um vertedor de soleira livre e largura $L = 20\,\mathrm{m}$. Use o coeficiente C calculado acima, e obtenha a vazão efluente $O(t)$ em função do tempo, de 1 em 1 minuto. Resolva o problema usando um método de diferenças finitas *de sua escolha* para a equação de balanço hídrico do reservatório,

$$\frac{\mathrm{d}S}{\mathrm{d}t} = I(t) - O(t).$$

SOLUÇÃO

As paredes do reservatório de acumulação são verticais:

$$S = AH.$$

No instante inicial, o reservatório está vazio:

$$S(0) = H(0) = 0.$$

Isso nos dá a condição inicial do problema.

A vazão afluente é "triangular", com duas retas cujas equações são facilmente obtidas:

$$I(t) = \begin{cases} t/360, & 0 \le t \le 3600, \\ 50/3 - t/540 & 3600 < t \le 9000. \end{cases}$$

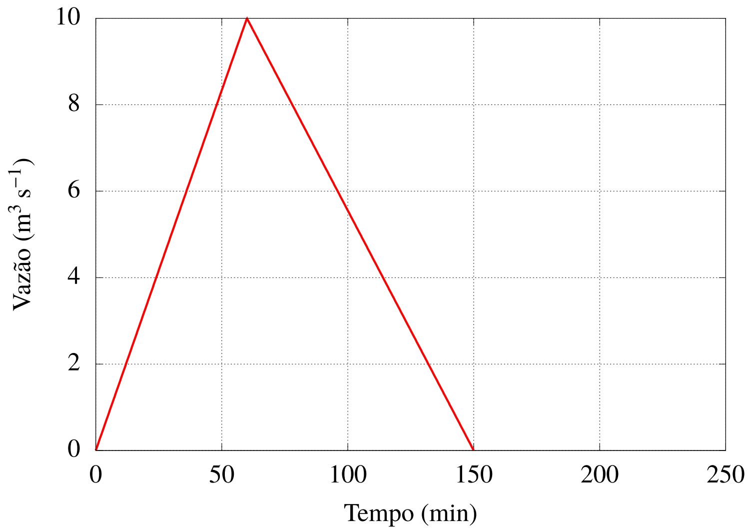


Figura 1: Cheia afluente a um reservatório de acumulação urbano.

Note que nós já convertemos as equações do gráfico da figura 1 para t em segundos. A equação diferencial que temos que resolver é

$$
\begin{aligned}
\frac{\mathrm{d}S}{\mathrm{d}t} &= I(t) - O(t),\\
S &= AH,\\
O(t) &= \alpha\sqrt{g}L[H(t)]^{3/2},\\
\frac{\mathrm{d}[AH]}{\mathrm{d}t} + \alpha\sqrt{g}L[H(t)]^{3/2} &= I(t),\\
\frac{\mathrm{d}H}{\mathrm{d}t} + \left[\frac{\alpha\sqrt{g}L}{A}\right] H^{3/2} &= \frac{I(t)}{A},\\
\frac{\mathrm{d}H}{\mathrm{d}t} + bH^{3/2} &= f(t),\\
b &= \frac{\alpha\sqrt{g}L}{A},\\
f(t) &= \frac{I(t)}{A}.
\end{aligned}
$$

7.1 Com um esquema de 1ª ordem

A forma mais simples de resolver numericamente a equação diferencial deste problema é utilizar um esquema de diferenças finitas explícito de 1ª ordem:

$$
\begin{aligned}
\frac{H_{n+1} - H_n}{\Delta t} + bH_n^{3/2} &= f(t_n);\\
H_{n+1} - H_n + b\Delta t H_n^{3/2} &= f(t_n)\Delta t;\\
H_{n+1} &= H_n - b\Delta t H_n^{3/2} + f(t_n)\Delta t;\\
H_{n+1} &= H_n + \Delta t\left[f(t_n) - bH_n^{3/2}\right].
\end{aligned}
$$

O programa de computador `rout01.py` foi escrito para resolver o problema, e é mostrado na listagem 4.

Listing 4: `rout01.py` — Propagação de cheia com um método explícito.

```
#!/home/nldias/miniconda3/bin/python3
#-*- coding: iso-8859-1 -*-
#-------------------------------------------------------------------
# rout01: propagação de cheia em um reservatório de acumulação de
# cheias com um esquema explícito
#
# Nelson Luís Dias
# 2020-07-05T12:26:50
# -------------------------------------------------------------------
from math import sqrt
# -------------------------------------------------------------------
# constantes do problema
# -------------------------------------------------------------------
A = 100000.0                  # área horizontal do reservatório
alfa = 0.493                  # constante universal para um vertedor
g = 9.81                      # aceleração da gravidade
L = 20.0                      # largura da soleira
b = alfa * sqrt(g) * L / A    # cte da eq diferencial
# -------------------------------------------------------------------
# hidrógrafa afluente (m3/s/m2)
# -------------------------------------------------------------------
def f(t):
   if t <= 3600:
      return (t/360.0)/A
   elif t <= 9000:
      return (50.0/3.0 - t/540.0)/A
   else :
      return 0.0
   pass
pass
# -------------------------------------------------------------------
# tudo pronto para resolver?
# -------------------------------------------------------------------
fou = open('rout01.out','wt')
told = 0
Iold = 0.0
Hold = 0.0                    # altura inicial
Oold = 0.0                    # hidrógrafa efluente inicial
fou.write('␣␣␣Tempo␣(s)␣I(t)␣(m3/s)␣O(t)␣(m3/s)\n')
fou.write('␣␣␣␣%8d␣␣␣␣%8.2f␣␣␣␣%8.2f\n' % (told,Iold,Oold))
# -------------------------------------------------------------------
# aplicação do método explícito. note que told, tnew e deltat são
# variáveis inteiras.
# -------------------------------------------------------------------
deltat = 60                   # passo de tempo de um minuto
while told < 36000 :
   tnew = told + deltat
   Inew = A*f(tnew)
   Hnew = Hold + deltat*( f(told) - b*Hold**1.5)
   Onew = b*A*Hnew**1.5
# -------------------------------------------------------------------
# imprime esta linha de resultados
# -------------------------------------------------------------------
   fou.write('␣␣␣␣%8.2f␣␣␣␣%8.2f␣␣␣␣%8.2f\n' % (tnew,Inew,Onew))
   told = tnew
   Hold = Hnew
pass
fou.close()
```

Graficamente, a figura 2 mostra o resultado da simulação.

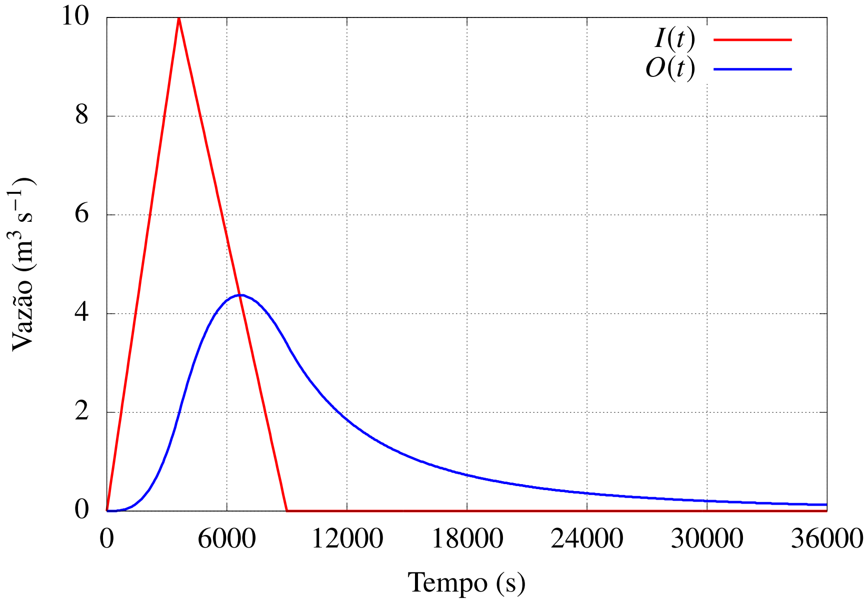


Figura 2: Simulação de um reservatório de acumulação de cheias com um método explícito.

7.2 Com um esquema de 4ª ordem (Runge-Kutta)

O mesmo problema também pode ser resolvido facilmente pelo método de Runge-Kutta de 4ª ordem (Dias, 2020). O programa `rout02.py` implementa essa solução alterntiva.

Inicialmente, colocamos a equação diferencial em uma forma reconhecível pelo método de Runge-Kutta:

$$\frac{\mathrm{d}H}{\mathrm{d}t} = F(t,H) = f(t) - bH^{3/2}.$$

Com isso, é muito simples implementar $F(t,H)$ (que denominamos `FF` no programa `rout02.py`). O programa completo é mostrado na listagem 5.

Listing 5: `rout02.py` — Propagação de cheia com o método de Runge-Kutta.

```
#!/home/nldias/miniconda3/bin/python3
# -*- coding: iso-8859-1 -*-
#-------------------------------------------------------------------
# rout02: propagação de cheia em um reservatório de acumulação de
# cheias com um método de Runge-Kutta de 4a ordem
#
# Nelson Luís Dias
# 2020-07-05T13:41:13
# -------------------------------------------------------------------
from math import sqrt
# -------------------------------------------------------------------
# constantes do problema
# -------------------------------------------------------------------
A = 100000.0                  # área horizontal do reservatório
alfa = 0.493                  # constante universal para um vertedor
g = 9.81                      # aceleração da gravidade
L = 20.0                      # largura da soleira
b = alfa * sqrt(g) * L / A    # cte da eq diferencial
# -------------------------------------------------------------------
# hidrógrafa afluente (m3/s/m2)
```

```
# ----------------------------------------------------------------------
def f(t):
   if t <= 3600:
      return (t/360.0)/A
   elif t <= 9000:
      return (50.0/3.0 - t/540.0)/A
   else :
      return 0.0
   pass
pass
def FF(t,H):
   return f(t) - b*H**1.5
pass
# ----------------------------------------------------------------------
# método de Runge-Kutta
# ----------------------------------------------------------------------
def rk4(t,H,deltat,FF):
   '''
   rk4 implementa um passo do método de Runge-Kutta de ordem 4
   '''
   k1 = deltat*FF(t,H)
   k2 = deltat*FF(t+deltat/2,H+k1/2)
   k3 = deltat*FF(t+deltat/2,H+k2/2)
   k4 = deltat*FF(t+deltat,H+k3)
   Hn = H + k1/6.0 + k2/3.0 + k3/3.0 + k4/6.0
   return Hn
pass
# ----------------------------------------------------------------------
# Propaga a cheia usando o método de Runge-Kutta de ordem 4
# ----------------------------------------------------------------------
deltat = 60.0                          # passo em t
t = [0.0]                              # t inicial
H = [0.0]                              # H inicial
NN = int(36000.0/deltat)               # número de passos
for n in range(0,NN):                  # loop da solução numérica
   tn = (n+1)*deltat                   # novo t
   Hn = rk4(t[n],H[n],deltat,FF)       # novo H
   t.append(tn)                        # adiciona t à lista
   H.append(Hn)                        # adiciona H à lista
pass
fou = open('rout02.out','wt')
fou.write('␣␣␣Tempo␣(s)␣I(t)␣(m3/s)␣O(t)␣(m3/s)\n')
for n in range(0,NN+1):            # imprime no arquivo de saída
   fou.write('␣␣␣␣%8d␣␣␣␣%8.2f␣␣␣␣%8.2f\n' % (t[n],A*f(t[n]),b*A*H[n]**1.5))
pass
fou.close()
```

Graficamente, a figura 3 mostra o resultado da simulação.

Finalmente, nós comparamos as duas soluções na figura 4. Como podemos ver, embora o método de Runge-Kutta seja teoricamente muito mais acurado, o intervalo de tempo de simulação $\Delta t = 60\,\mathrm{s}$ é suficientemente pequeno para que o método explícito produza um bom resultado, praticamente igual ao obtido pelo método de Runge-Kutta.

Referências

Chow, V. T., Maidment, D. R., e Mays, L. W. (1988). *Applied Hydrology*. McGraw-Hill, New York.

Dias, N. L. (2020). *Uma Introdução aos Métodos Matemáticos para Engenharia*. Edição do Autor, Curitiba, 2nd edição. Available at https://nldias.github.io/pdf/matappa-2ed.pdf.

Penman, H. (1948). Natural evaporation from open water, bare soil and grass. *P Roy Soc London*, A(193):120–146.

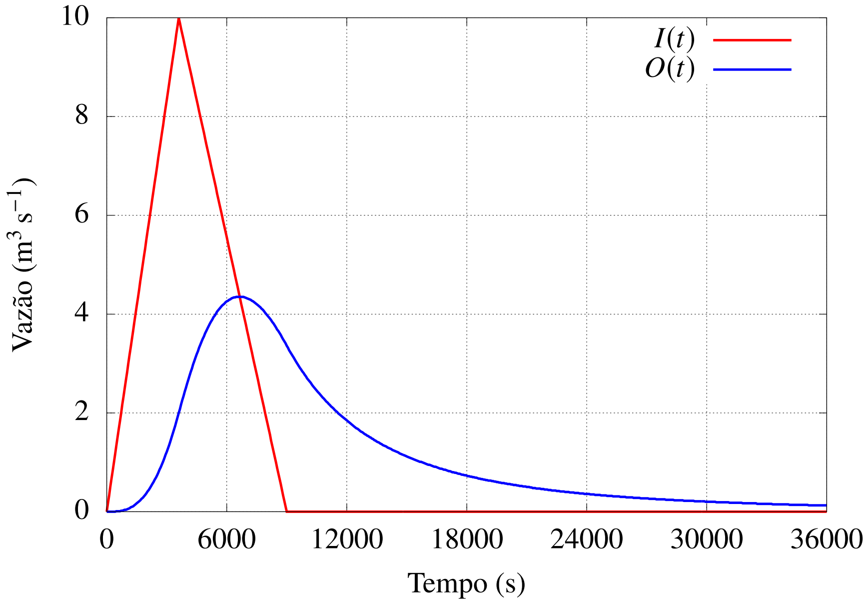


Figura 3: Simulação de um reservatório de acumulação de cheias com o método de Runge-Kutta.

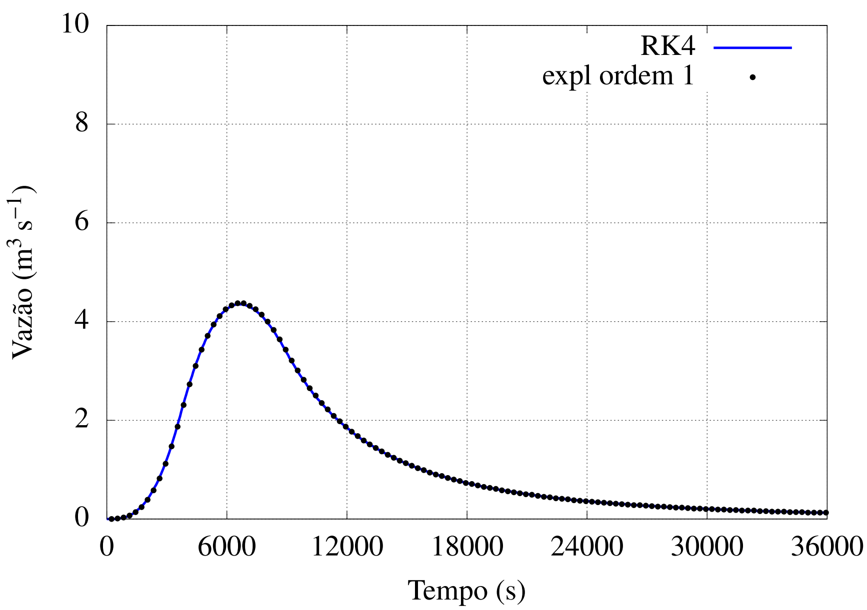


Figura 4: Comparação entre os métodos explícito (ordem 1) (pontos pretos) e de Runge-Kutta (linha azul) para a simulação de um reservatório de acumulação de cheias.

Stelling, E. (1882). Über die Abhängigkeit der Verdunstung des Vassers von seiner Temperatur und von der Feuchtigkeit und Bewegung der Luft (vorgelet 1881). *Repertorium für Meteorologie, Kaiserliche Akademie der Wissenschaften, St. Petersburg*, 8(3):1–49.