RESCIENCE C

Replication / Physics

[Re] Reproduction of Step width enhancement in a pulse-driven Josephson junction

Sabino Maggi[1, ID]

[1]National Research Council, Institute of Atmospheric Pollution Research, CNR-IIA, Bari, Italy

Edited by
(Editor)

Received
–

Published
–

DOI
–

Abstract When I read about the Ten Years Challenge, it was a matter of seconds to take the plunge and decide to partecipate with the replication of a work of a quarter of century ago about the numerical simulation of a Josephson junction driven by a train of microwave pulses. I was fairly dubious to be able to replicate the results of that work, but in the end the effort resulted much easier than I initially thought. The paper describes the tools and tricks used for the replication and the lessons learned from this work. Among them, keeping good and updated documentation is paramount. Likewise, the use of tools and languages that do not fall into oblivion after one or two iterations.

Introduction

[++ Simulate the junction behavior... ++]
[++ mcphase calculates both the IV curve and the phase path (?) of the junction (outside the scope of the present paper) but only for a single value of α_J ++]
A Josephson junction is a quantum mechanical device composed of two superconducting electrodes separated by a weak link [1]. For currents lower than a critical value I_C, coupled electrons (Cooper pairs) can cross the weak link without any potential difference (dc Josephson effect). When the current is increased above I_C, single electrons originated by the breakup of Cooper pairs begin to traverse the weak link. The potential difference V between the two superconducting films becomes $\neq 0$ and a state is reached where the junction behaves as a resistance.
In modern Josephson junctions the weak link usually has the form of a thin insulating tunnel barrier (SIS junction) [2], a normal metal film (SNS junction) [3] or a physical nanoconstriction (ScS junction) [4, 5]. Josephson junctions have found wide usage in several research fields, for example as building blocks for RSFQ digital electronics or quantum computers [6], or as radiation detectors and very sensitive magnetometers (SQUIDs) [7, 8]. But the most successful application of Josephson junctions is surely in voltage metrology. A microwave radiation of frequency f can phase lock the junction current, producing the so-called Shapiro-steps, i.e., current steps at the quantized voltages V_n,

$$V_n = n\frac{h}{2e}f, \quad n = 1, 2, \ldots \tag{1}$$

where h and e are the Plank constant and electron charge, respectively. The ac Josephson effect is at the basis of the quantum voltage standard that from 1987 replaced the previous voltage standard based on Weston cells.
Besides its practical applications, the Josephson junction is important from a physical point of view because it has been the first device showing quantum mechanical effects on a macroscopic scale.
An important research topic at the beginning of the '90s was related to finding ways to increase the amplitude of the current steps induced by the microwave radiation. In fact,

Correspondence should be addressed to Sabino Maggi (sabino.maggi@cnr.it)
The authors have declared that no competing interests exists.

the stability of the lock between the phase of the junction and the applied microwave bias – and therefore its insensitivity to noise events which might switch the junction from one quantized level to another, which is a crucial problem for voltage standard applications – is strongly dependent on the step amplitude [9].
To increase the amplitude of the current steps, a non-sinusoidal microwave radiation may be used. In 1990 Monaco showed that, in the limit of a voltage-biased junction, adding together two phased microwaves of frequencies f and $2f$ produces rf-induced current steps whose normalized amplitudes are larger than those observed with a standard sinusoidal radiation [10]. Experiments on the so-called "biharmonic drive" readily confirmed these conclusions, albeit with some limitations due to the fact that the junctions could not be properly considered as being voltage biased [11, 12]. Extending further the idea, Monaco showed that for a microwave radiation composed of a train of periodic delta functions the rf-induced current steps of a voltage-biased junction could be as large as the critical current I_C.
However, in practice a voltage bias configuration does not properly model a real Josephson junction, which usually should be considered as current biased. Further, pulse train composed of delta functions is only theoretical and cannot be reproduced in actual experiments. This led to the idea to investigate what happened to a current-biased Josephson junction irradiated by a more realistic pulsed microwave signal [13, 14]. The reproduction of this investigation is the object of the present work.

2 Computational context

A first attempt to solve this problem was made by using an electronic analog simulator of a Josephson junction [15], that could compute the relation between the applied current and the voltage ($I-V$ characteristic) of a current-biased junction, in the framework of the Stewart-McCumber RSJ junction model [16, 17]. The analog simulator was very fast and simple to use, and could produce in more seconds beautiful plots of the $I-V$ characteristics of the junction as a function of the simulated microwave signal on a Hewlett Packard 7475A2 pen plotter. [1]
However, even if the electronic simulation was extremely fast, analysis of the results required to measure by hand the amplitude of the rf-induced steps visible on each $I-V$ curve of the paper plots, a long and error-prone work.
I then decided to write a Fortran program to solve numerically the nonlinear second-order differential equation that models the Josephson junction [16, 17]. The idea was to simulate the junction behavior by calculating the $I-V$ characteristics of the junction as a function of the pulse amplitude α_{rf} for a given set of parameters characterizing the junction and the applied microwave signal, considering the three cases of standard sinusoidal drive, biharmonic drive and realistic pulsed drive. To ease comparison of the results, normalized units have been used throughout the calculations [16, 17]. The main parameters of the simulation were: the hysteresis parameter β, the frequency Ω and the width of the pulsed signal ρ. For a for a given set of junction parameters, around 100 different $I-V$ characteristics for increasing or decreasing values of α_{rf} were calculated.
The first versions of the Fortran program were compiled under DOS 6.22 with Microsoft Fortran 5.1 and run on what was then a state-of-the-art PC, probably a Compaq Deskpro 486 with a math coprocessor, shared among several users of the lab. However, after a few attempts, the limitations of a PC for such a task soon become evident. A new simulation started automatically each evening and took the entire night to be completed. Unfortunately, people still using the PC late in the evening often inadvertently stopped the background process or simply shut the machine down without checking if a job was already running. At the end, I could run a full simulation only every two or three days.

[1] Unfortunately, after 25 years and two relocations, I could not manage to find pictures of the simulator nor the original HP 7475A plots.

After a few weeks of these mostly unsuccessful attempts, the sysadmin of another research group proposed me to use four DEC workstations running ULTRIX (or possibly Digital UNIX), for my own simulations.[2] The machines were heavily used by his group during working hours but sat mostly idle overnight. If I could manage to finish my runs before the start of the new work day, I was allowed to use this idle time for my simulations. My colleague gave me a quick crash course on Unix and I was ready to go. Porting my Fortran program from Microsoft Fortran 5.1 to ULTRIX was a snap, and I also quickly learnt how to use FTP to transfer the configuration files and the results of the simulation back and forth from the DEC machines to my Compaq 386 notebook, that was also my desktop computer.

Now at the beginning of each day I had four different sets of files coming from the four available DEC machines. Three of them were made by irradiating the junction with a pulsed drive with decreasing values of ρ (i.e., the normalized width of the pulse train), while keeping constant β and Ω. The fourth simulation was made by irradiating the junction with a standard sinusoidal signal and keeping everything else equal. This last simulation was used as a reference, to compare the results obtained with the standard sinusoidal microwave signal with those obtained with progressively shorter pulses. Each night I changed the values of β or Ω to study the effect of these parameters on the junction behavior.

The real problem was again how to analyze all this data. A manual analysis like that needed by the electronic simulator was out of consideration, and I decided to to write another program, using the recently released Microsoft Visual Basic 1.0 for Windows, that calculated from the nightly simulation results the size of all the rf-induced steps visible on the $I - V$ characteristics as a function of the rf bias α_{rf}. Now I could transfer by FTP the files containing the nightly results from the DEC workstations to my desktop computer and run the Visual Basic program on each data set to quickly get a summary file containing the size of the quantized current steps for the reference sinusoidal drive and for the three simulated junctions subjected to progressively shorter current pulses. The results of months of calculations were summarized in a paper published in the Journal of Applied Physics [13].

3 Digging into code

I am the type of person who likes organisation, and I try keep all my past projects on my main workstation. Thus, finding the original source files of this project was only a question of locating the directory where the project was stored. Problems started to arise when I looked at the different files. The whole project was scattered into several directories, each containing many files with widely different names and dates. At first, trying to find an order in that chaos seemed impossible. Normally I would have found many notebooks full of detailed handwritten notes. Regrettably, a couple of years ago most of my work notebooks were damaged by a water leak in the basement, and could not be recovered. So the only option was to check the files one by one. Luckily modern operating system greatly ease the burden of dealing with data files; in particular, Quick Look for macOS is an exceptional tool to quickly browse through hundreds of files at the touch of the spacebar.

After a thorough inspection of the whole project I recalled that: (1) my first attempts with the numerical simulations used the more accurate McDonald-Johnson junction model [18], and that I later switched to the simplified Stewart-McCumber RSJ model because it was much faster and efficient in calculating the junction behavior [16, 17]; (2) file names tried to reflect what the programs actually did. I could have in the same directory a file ending with a "t" that provided a textual output and the same files ending with a "g" that

[2] The now defunct Digital Equipment Corporation was one of the leading computer companies of the time.
[3] At least within the limits of DOS 8+3 naming scheme, only 8 characters for the file name and 3 characters for the extension.

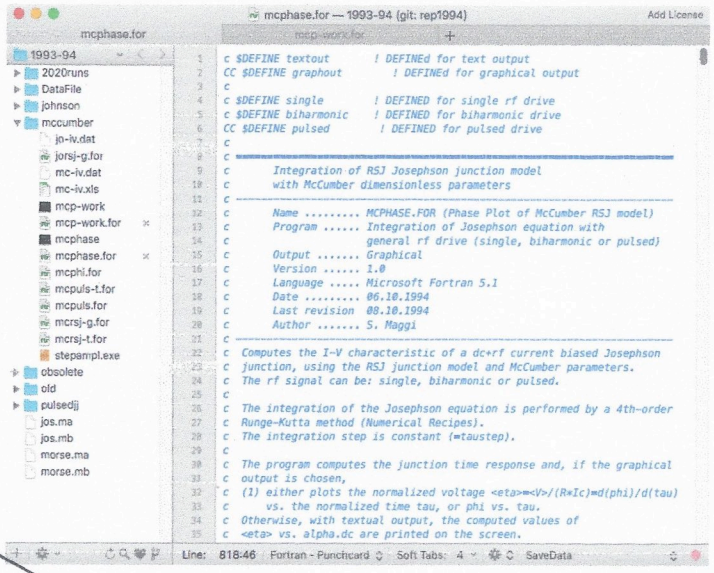

Figure 1. Header of one of the Fortran source files. The left sidebar shows the directory structure of the project.

gave a graphical output. Having many files differing only for a couple of DEFINEs might seem senseless today, when comfortable graphical interfaces and ultra-fast text editors with support of regular expressions allow to change a file in just a few seconds, but at that time it was probably the fastest, albeit very inefficient, way of working with source code; (3) luckily all source files had an header containing detailed notes about the type of program, the compiler, the type of output, and what probably mattered more, the dates of first and last revision of the source file (Fig. 1), and that eased much the analysis of the different versions of the Fortran source files; (4) the initial versions of the Fortran programs were monolithic, a single source file contained the whole code that consisted of around 1.000 lines of Fortran. Only at a later time, better computing practices led me to divide the monolithic code into multiple source files that were compiled and linked together using a Makefile.

An invaluable tool to analyze the different versions of the source files was Meld, an open source tool for all major operating systems that can perform a two- and even a three-way comparison of files and directories. Using Meld I quickly realized that the two main source files were mcphase.for and mcp-work.for, both located in the mccumber/ directory (Fig. 2).

The first program, mcphase, simulated the behavior of the junction for a single value of α_{rf}, read from the input configuration file iv.dat, saving the $I - V$ characteristic of the junction and its phase portrait (i.e., the relation between the phase and its time derivative, the latter being proportional to the junction voltage V) in the output file iv.out. Multiple calculations with several different value of α_{rf} could be performed by using a simple DOS batch file that basically choose one by one the configuration files containing the desired values of α_{rf}, renamed them to iv.dat, run mcphase and at the end renamed iv.out using a consistent naming scheme. I don't recall why I choose this approach, but it was clearly very inefficient and not prone, as it required to prepare each day a long series of configuration files that differed only by the value of α_{rf}, and to update accordingly the DOS batch file that controlled the night calculations (Fig. 3).

The second program, mcp-work.for was an improved version that could automatically cycle across a set of values of α_{rf}, producing a different output file for each value of α_{rf}. However, to simplify further automatic analysis, it left out the phase portrait [4]

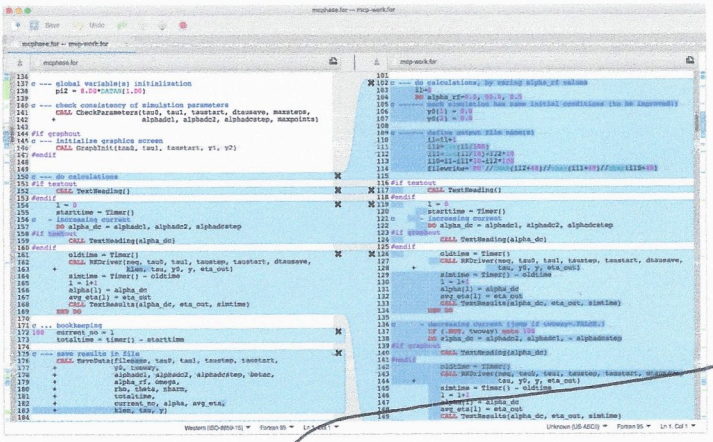

Figure 2. Comparison of the main loop of mcphase.for and mcp-work.for with Meld.

MIGLIORARE/TOGLIERE ++] It was clear that this was the program ported to the DEC workstations. Unfortunately, I could not find the actual source file used on these machines, probably because while performing the porting from DOS/Microsoft Fortran 5.1 to ULTRIX I worked directly on the workstations and never thought to copy back these source files since they could not work on my PCs. Luckily, Fortran is a very stable language and porting again both mcphase.for and mcp-work.for so that they could work on a modern machine was a snap. As for the Microsoft Visual Basic 1.0 code, there were only two versions of the Visual Basic analysis programs and the differences between them were minimal. Since both programs gave exactly the same results, I decided to stick with the initial version that had a working (precompiled) binary.

4 Porting Microsoft Fortran to modern Unix

Porting mcphase.for and mcp-work.for to the XXI century so that they could be compiled with the modern open source and multiplatform Fortran gfortran compiler was very easy, thanks to the stability of the language across different versions and platforms. Essentialy only a few tweaks to the source code were needed.
All work has been done on macOS, which is essentially BSD Unix with a more appealing graphical interface, but could be easily repeated on any modern Unix-like operating system such as Linux, or even on Windows, with the support of either the Windows Subsystem for Linux (for Windows 10) or of Cygwin (for earlier versions of the operating system).

4.1 Preprocessor directives

For reasons that go beyond my understanding Microsoft Fortran 5.1 did not use standard preprocessor directives, such as those supported by cpp or fpp, [19] but used a a slightly different proprietary syntax (Fig. 4). Luckily, It was very easy to comment out all the $DEFINE directives in the header section which specified which sections of the code to compile, and to replace $DEFINE blocks with standard cpp blocks throughout the code. Microsoft Fortran 5.1 (Fig. 4) [++ SUPPLEMENTARY? ++]. The right directives are chosen now at compile-time. For example, the following command [4]

[4] The $ symbol prepended to this and all following terminal commands represents the prompt of the command interpreter and is not part of the command.

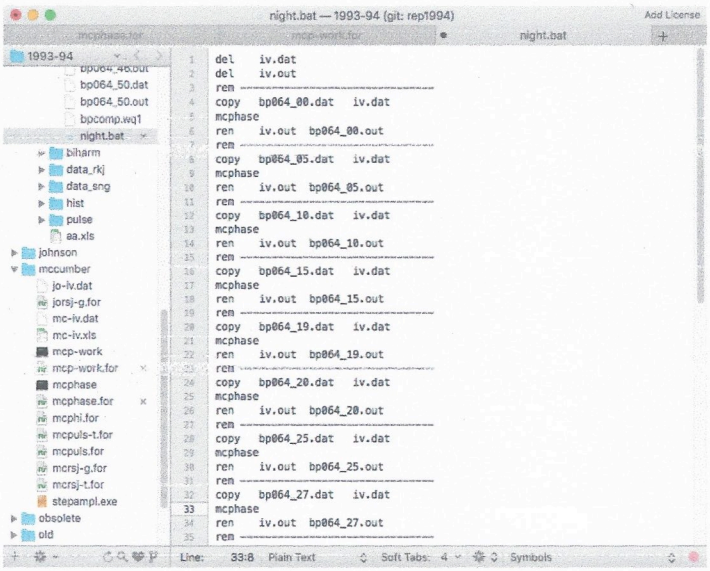

Figure 3. DOS batch file used by mcphase.for to perform multiple simulations with different values of α_{rf}.

```
$ gfortran -cpp -Dtextout -Dsingle -o mcp-work mcp-work.for
```

runs the cpp preprocessor before compilation of mcp-work.for with gfortran, selecting the sections of the code that produce a textual output (-Dtextout) and performing the calculations with the single (sinusoidal) rf bias (-Dsingle).

4.2 Filenames

Ancient Fortran compilers such as Microsoft Fortran 5.1 did not support dynamic memory allocation at runtime and required programmers to use fixed-length arrays and strings. Strings were used rather sparingly in Fortran code, so that was not a big deal. With an exception. My code defined the basename of all files as a 50-byte long string,

```
CHARACTER*50   filename
```

attaching a proper extension to differentiate between the input files containing the simulation parameters and the output file with the results of the calculations.
Under DOS that was not a problem, as DOS truncates file names to only 8 characters plus

(a)

```
$DEFINE textout        ! DEFINEd for text output
c $DEFINE graphout     ! DEFINEd for graphical output

c $DEFINE single       ! DEFINED for single rf drive
c $DEFINE biharmonic   ! DEFINED for biharmonic drive
$DEFINE pulsed         ! DEFINED for pulsed drive
...
$if defined (graphout)
    ...
$endif
...
...
$if defined (textout)
    ...
$endif
...
$if defined (pulsed)
    ...
$endif
```

(b)

```
CC $DEFINE textout     ! DEFINEd for text output
c $DEFINE graphout     ! DEFINEd for graphical output

c $DEFINE single       ! DEFINED for single rf drive
c $DEFINE biharmonic   ! DEFINED for biharmonic drive
CC $DEFINE pulsed      ! DEFINED for pulsed drive
...
#if graphout
    ...
#endif
...
...
#if textout
    ...
#endif
...
#if pulsed
    ...
#endif
```

Figure 4. Preprocessor directives in (a) Microsoft Fortran 5.1, (b) modern cpp preprocessor.

3 characters for the extension, and all excess characters were simply ignored. But under Unix file names have no real limitations,[5] and having these 50 character-long filenames, mostly composed by blank characters, was ugly and complicated file management, in particular when using the command line interface. The solution is simple, as Fortran now has the TRIM function that removes all trailing blanks from a string. Whenever a file is opened for reading or for writing, it is sufficient to apply on the fly TRIM() to the filename variable, as shown here,

```
OPEN (UNIT = 10, FILE = TRIM(filename)//'.dat', STATUS = 'OLD')
```

to remove all extra blanks from the name of the file.

4.3 Edit descriptors

Microsoft Fortran 5.1 used the backslash \ edit descriptor to prevent the addition of a line break at the end of a WRITE instruction. Modern Fortran compilers do not support this non-standard edit descriptor and return an error. To avoid this problem, it is sufficient to remove the backslash edit descriptor from all WRITE instructions that include it.

4.4 Date and time

Microsoft Fortran 5.1 had two separate intrinsic subroutines to return the current date and time. In particular, `CALL GETDAT(iyr, imon, iday)`, saved the date in the two-byte integer variables iyr, imon and iday, while `CALL GETTIM(ihr, imin, imin, i100th )` did the same for the current time, saving the return values in the integer variables ihr, imin, imin and i100th. The meaning of each returned variable should be self-explanatory. Modern Fortran supports the single subroutine `CALL DATE_AND_TIME([DATE], [TIME], [ZONE], [VALUES])`, where all arguments are optional and can be specified by their dummy names (i.e., how Fortran calls the keyword arguments of a function call). In particular, DATE, TIME and ZONE are character variables, while VALUES is a one-dimensional array of 8 integers, where VALUES(1:3) corresponds to the year, month and day of the month, VALUES(4) is the time difference (in minutes) with UTC, and VALUES(5:8) are the hour, minute, second and milliseconds, respectively.
To minimize changes to the original source code, the original calls to the `GETDAT` and `GETTIM` subroutines,

```
INTEGER*2       iyr, imon, iday
INTEGER*2       ihr, imin, isec, dummy
...
CALL GETDAT(iyr, imon, iday)
CALL GETTIM(ihr, imin, isec, dummy)
```

were translated to a single call to `DATE_AND_TIME`, assigning the elements of the returned array of VALUES to the integer variables of the original code,

```
character*8     date
character*10    time
character*5     zone
integer         values(8)
...
call date_and_time(date, time, zone, values)

iyr  = values(1)
imon = values(2)
iday = values(3)
ihr  = values(5)
imin = values(6)
isec = values(7)
```

[5] To be precise, Unix allows 255 characters for the filename and 4096 characters for the path.

4.5 Compilation with gfortran

All these fixes were enough to compile correctly mcphase.for with gfortran. As for mcp-work.for, basically it required the same corrections, with one rather strange addition. As noted above, mcp-work.for calculates the $I-V$ characteristics of the simulated junction for several different values of α_{rf}, producing one output file for each $I-V$ curve. The code to define the output file names was very convoluted,

```
c --- do calculations, by varing alpha_rf values
      il=0
      DO alpha_rf=0.0, 50.0, 0.5
      ...
c ------ define output file name(s)
         il=il+1
         il2=Int(il/100)
         il1=int(il/10)-il2*10
         il0=il-il1*10-il2*100
         filewrite='PU'//CHAR(il2+47)//char(il1+47)//char(il0+47)
      ...
      END DO          ! repeat alpha_rf DO cycle
```

and used the integer variable il to count the cycle number, while the three integers il2, il1 and il0 contained the hundreds, tens and units digits of il, respectively. These integers are converted to the corresponding ASCII characters with the function CHAR, where ASCII code 48 corresponds to the 0 symbol and ASCII code 57 corresponds to 9. The name of the output file filewrite was built by concatenating a trailing constant string to the three ASCII characters with the (//) operator.

I have no idea why I decided to build the name the output files in such a complex way, while it would have been much easier to use the value of α_{rf} to name each output file. In any case, it did not work under gfortran, preventing proper compilation of the code. After some inspection and testing, it was apparent that the source of the problem lied in the number 47 added to three integer variables before converting them to ASCII characters, that should be replaced by 48, so that the line defining the filewrite variable becomes `filewrite='PU'//CHAR(il2+48)//char(il1+48)//char(il0+48)`.

With this change mcp-work.for could compile flawlessly under gfortran. What is really puzzling is how the original line could work in Microsoft Fortran 5.1.

5 Visual Basic code

No attempt was made to make the original Visual Basic 1.0 code run a modern computer. Visual Basic is a dead language and long since has been replaced by Visual Basic .NET, which shares only the name with its forefather. The only viable option from the beginning was to use an emulator to rebuild the original development environment. For this task I preferred to use Parallels Desktop, but the open source multi-platform VirtualBox emulator can be used equally well.

I created a new empty virtual machine with minimal *hardware* requirements and installed in sequence DOS 6.22, Windows 3.11 and Visual Basic 1.0 (Fig. 5). While I was at it, and although I had already solved the Fortran part of the project, I also decided to install Microsoft Fortran 5.1 for DOS, to try to reproduce as much as possible the original work environment.

All software packages were downloaded from the WinWorld web site, an invaluable resource for recovering old software packages. The legitimacy of installing proprietary software in an emulator, even after so many years, might be questionable. But at the time I had regular licenses for all the above mentioned software and I believe to be at least morally authorized to continue to use those packages.

The installation of these aging software packages in an emulator is very close to how it was done back then. The packages were originally distributed on several floppy disks, which had to be swapped whenever the installer required a new disk. Today, the floppy disks are replaced by virtual file images and "swapping" disks is not done mechanically but requires to select a menu option in the emulator.

[++ DOPO? ++] The default screen resolution of Windows 3.11 when installed in the emulator was a mere 640 × 480 pixels, woefully meager by today's standards. However, as I used the virtual environment almost exclusively to run the Visual Basic program, I didn't bother to install the drivers that could increase the screen resolution to a more comfortable 800 × 600 or 1024 × 768 pixel resolution (Fig. 5). [++ DOPO? ++]

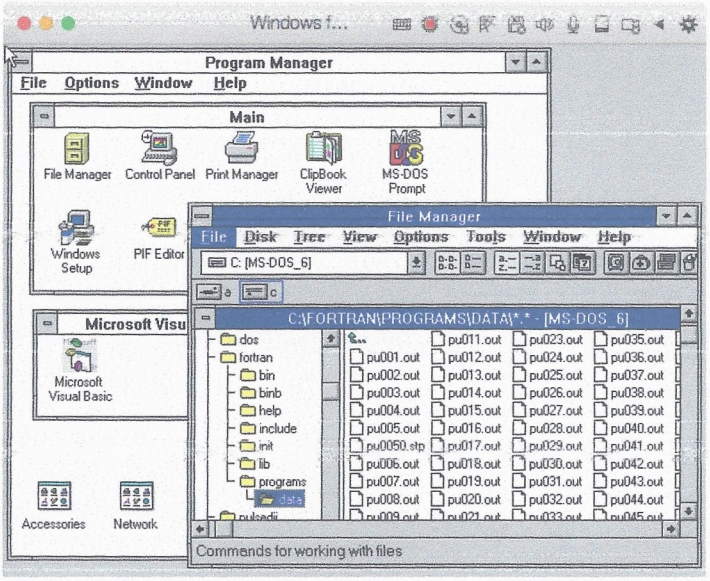

Figure 5. The Windows 3.11 desktop as shown in the Parallels emulator.

Using the emulator required to transfer the source and binary Visual Basic files to the emulated DOS/Windows system. I excluded the possibility of setting up Windows 3.11 networking in the emulator, in order to make Windows 3.11 communicate with the host OS through the network. It probably can be done, but it was much easier to transfer all needed files using again a virtual floppy disk, mounted alternatively (and exclusively) on macOS or on Windows 3.11.

A new virtual floppy disk data.img can be easily created with the command-line utility dd available on macOS or Linux, `dd if=/dev/zero of=data.img bs=1440k count=1`. After creation, the virtual floppy disk must be mounted in the emulator and formatted under DOS or Windows 3.11 in the original MS-DOS FAT file system.

The same virtual floppy disk was used to copy the source code of mcphase.for and mcpwork.for and the mc-iv.dat configuration file to the virtual machine. [++, transferring them to the Microsoft Fortran 5.1 standard project directories.++]

This step completed the preparation of the development environment, now it was time to test how all this behaved.

6 Running the programs

To prevent cluttering the mccumber/ directory containing the Fortran source files (Section 3) with the output data files produced by the simulations, I created a new direc-

tory, 2020runs/, in the main project folder, where I copied the mc-iv.dat configuration file needed to start the simulation (Fig. 1).

Running mcphase.for is easy. In summary, all I had to do was to compile mcphase.for with the right directives, switch to the 2020runs/ directory and run the mcphase executable from there, as summarized below,

```
$ gfortran -cpp -Dtextout -Dsingle -o mcphase mcphase.for
$ cd ../2020runs/
$ ../mccumber/mcphase
```

obtaining a file mc-iv.out that contains the $I - V$ characteristic and the phase portrait (the relation between the phase and its time derivative, the latter being proportional to the junction voltage V) of the junction for the set of parameters defined in mc-iv.dat. The calculation takes only a couple of seconds to complete on a recent computer, and without microwave radiation ($\alpha_{rf} = 0$), produces the well-known non-hysteretic $I - V$ characteristic of the RSJ model (Fig. 6a), while for non-zero values of α_{rf} the staircase-like structure of the rf-induced current steps appear on the $I - V$ curves (Fig. 6b and Fig. 6c).

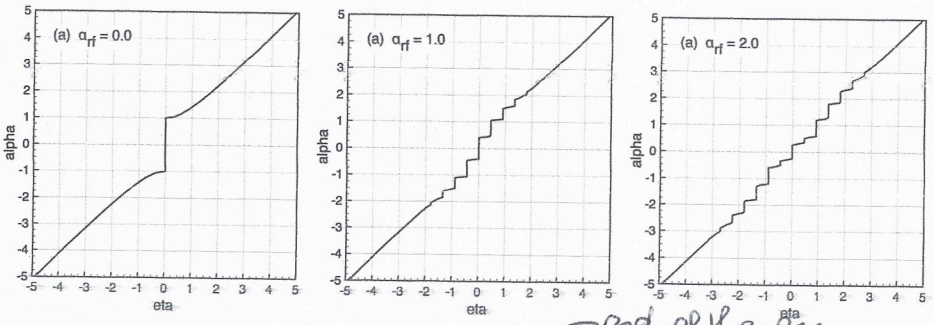

Figure 6. $I - V$ characteristics of a junction with $\beta_c = 0.01$, driven by a sinusoidal rf bias of frequency $\Omega = 0.45$ and (a) $\alpha_{rf} = 0.0$, (b) $\alpha_{rf} = 1.0$ and (c) $\alpha_{rf} = 2.0$. The rf-induced current steps are clearly visible when $\alpha_{rf} > 0$.

To simulate a pulsed rf bias, it is sufficient to compile mcphase.for with the -Dpulsed directive and rerun the simulation (Fig. 7). The $I - V$ characteristic without the rf bias is identical to that calculated with the sinusoidal signal (Fig. 7a), while the current steps induced by the pulsed drive are fewer and nearly as wide as the critical current for $\alpha_{rf} = 0.0$ (compare Figs. 7a and 7c).

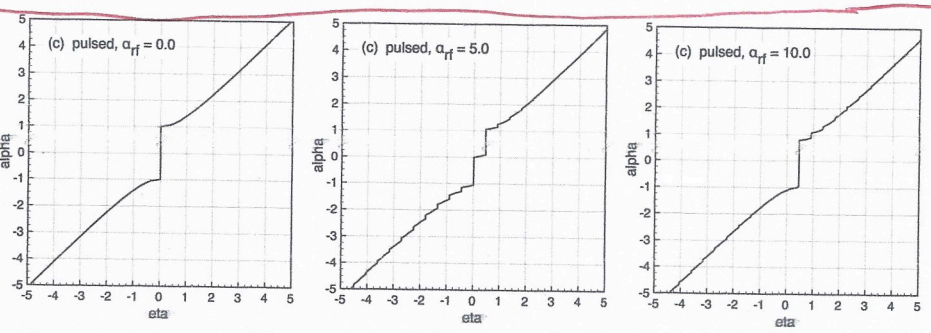

Figure 7. $I - V$ characteristics of a junction with $\beta_c = 0.01$, driven by a pulsed rf bias of frequency $\Omega = 0.45$ for: (a) $\alpha_{rf} = 0.0$, (b) $\alpha_{rf} = 5.0$ and (c) $\alpha_{rf} = 10.0$. For $\alpha_{rf} > 0$ the rf-induced steps are much larger than with the standard sinusoidal rf-drive.

The second Fortran program, mcp-work.for, is much more interesting. While mcphase.for can calculate the $I - V$ characteristic of the junction only for a single value of α_{rf} and

for increasing or decreasing values of the normalized current α_{dc}, mcp-work.for sweeps the normalized current α_{dc} in both directions to help visualize the possible hysteresis of the junction and calculates the $I - V$ characteristics for a range of values of α_{rf}, saving each curve in a separate output file. Unfortunately, the lower and upper limits and the step size of α_{rf} are hardcoded in the Fortran source code ad every change requires a recompilation of mcp-work.for, a minor hassle, as the compilation takes just a couple of seconds on a modern machine.

Running mcp-work.for is very similar to the previous case. First, it is necessary to compile mcp-work.for with the proper compiler directives, switch to the 2020runs/ directory and run the mcp-work executable from there. The whole process is summarized below for the standard sinusoidal drive,

```
$ gfortran -cpp -Dtextout -Dsingle -o mcp-work mcp-work.for
$ cd ../2020runs/
$ ../mccumber/mcp-work
```

The only modification needed to perform calculations using the pulsed drive is to change the -Dsingle directive to -Dpulsed,

```
$ gfortran -cpp -Dtextout -Dpulsed -o mcp-work mcp-work.for
$ cd ../2020runs/
$ ../mccumber/mcp-work
```

Also the names of the output files are hard-coded in the mcp-work.for in the filewrite variable and are conventionally composed by a two-letter prefix ("SI" for the single drive, "BI" for the biharmonic driva and "PU" for the pulsed drive) followed by a three-digit integer that represents the cycle number (Section 4.5).

On a modern machine the whole simulation with $100\alpha_{rf}$ steps takes around 2–4 minutes and most of the time is spent printing on the terminal the calculated $I-V$ characteristics for each value of α_{rf}. Such feedback was necessary at the time of the original calculation, as every new calculated point of the $I - V$ curves appeared on the screen after several tens of seconds, now the results sroll on the screen at a speed that makes them almost illegible. However, to keep as faithful as possible to the original project, I decided to continue to print data on the computer screen.

At the end of the run, the output files must be transferred to the Windows 3.11 virtual machine to be processed by stepampl, the Visual Basic application described in Section 5. However Windows 3.11 does not *understand* Unix line terminators and cannot read the output files without a preliminary conversion. This conversion can be easily done in the macOS Terminal with the following command

```
$ for f in $(ls *.out); do sed -i .bak s/$/$'\r'/ $f ; done
```

that changes the line terminators of the .out output files from the Unix format containing only a line-feed (LF) to the carriage return followed by a line-feed (CR-LF) format used by all versions of Microsoft Windows. [6]. The -i switch allows in-place conversion of each file, the original files are kept adding a .bak extension.

Now the output files in the proper Windows compatible format can be transferred onto the virtual floppy disk image. When the transfer is done, the floppy disk image is unmounted from the host operating system and mounted in the virtual machine, making it visible to Windows 3.11. The output files are copied to an empty directory of Windows 3.11 and the Visual Basic application is started, either by running the precompiled stepampl.exe application or by opening the Visual Basic project and running the program form there (Fig. 8), that calculates the size of the quantized current steps for each value of α_{rf} and saves the results in another text file with extension .STP (for steps) that could be transferred back to the host operating system via the virtual floppy disk image.

The original Visual Basic code had a hard coded limit set to a maximum number of 100 processed files. This limit was probably related to memory problems or simply to the

[6] Slightly different versions of the command can be found on the internet; the format used above is POSIX-compliant and should run on any Unix flavour

fact that it was impossible to run an overnight simulation with more than 100 different α_{rf} values.

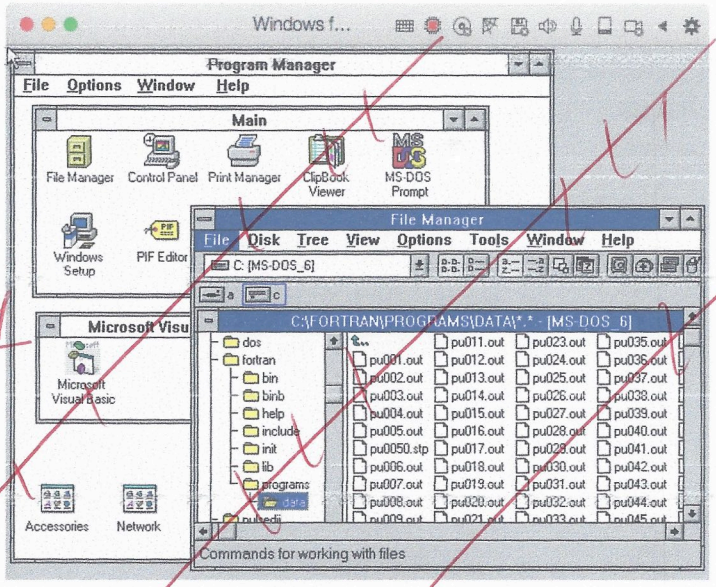

Figure 8. Windows 3.11 desktop showing the program stepampl that has just finished to process the data files.

At the time of writing the original paper [13], the whole process had to be repeated each night for a different set of input parameters and for a different kind of rf bias (single, biharmonic or pulsed). As noted above (Section 2).

Each night I used 3 of the available DEC machines to simulate in parallel the junction behavior with the pulsed drive, using 3 different values of the (normalized) width of the pulse signal, ρ, while keeping constant all the other junction parameters, such β and Ω. The only other difference in these 3 simulation was the range of variation of α_{rf}, which depended on the value of ρ (shorter pulses require a much larger intensity of the rf-signal to have the same effect on the junction). The fourth machine was reserved to simulate the reference junction behavior with the standard sinusoidal drive, using exactly the same set of junction parameters.

The original paper contained all the information needed to reproduce the results shown in the figures, without having to repeat the whole analysis from scratch. The parameters used in all runs were: hysteresis parameter $\beta = 0.01$, frequency of the sinusoidal or pulsed rf signal $\Omega = 0.45$, normalized current bias between $\alpha_{\mathrm{dc}} = -5.0$ and $\alpha_{\mathrm{dc}} = 5.0$, normalized integration time $\tau = 500$ with step $\Delta\tau = 0.01$.

The simulation with the sinusoidal drive was performed by varying the amplitude of the rf signal α_{rf} between 0.0 and 5.0, with a step $\Delta\alpha_{\mathrm{rf}} = 0.05$, while the three simulations with the pulsed drive were done using: (1) pulse width $\rho = 0.250$, $\alpha_{\mathrm{rf}} = 0.0 - 10.0$, $\Delta\alpha_{\mathrm{rf}} = 0.1$; (2) pulse width $\rho = 0.125$, $\alpha_{\mathrm{rf}} = 0.0 - 20.0$, $\Delta\alpha_{\mathrm{rf}} = 0.2$; (3) pulse width $\rho = 0.050$, $\alpha_{\mathrm{rf}} = 0.0 - 50.0$, $\Delta\alpha_{\mathrm{rf}} = 0.5$.

The resulting output and summary files were saved in separate folders in the 2020runs/ directory, named SINGLE/, PULS0250, PULS125, PULS0050 after their DOS counterparts.

To reproduce the first and second figure of [13] I made the only concession to modernity. instead of trying to recreate them with the plotting program used originally, probably Origin 2.0, I decided to write a couple of small R scripts that could automate the task. The results are shown in Fig. 9 and Fig. 10 and, as expected, are identical to those reported in the first two figures of [13]. Figure 3 of the original paper could be easily reproduced by plotting the maxima of the curves of Fig. 9, i.e., Δi_n vs. α_{rf}, for the four dif-

ferent cases considered here. Note that the large vertical steps on the rightmost curves of Fig. 10 era the first rf-induced steps, that can be nearly as large as the critical current I_C without rf bias visible in the first plot on the left.
Similar considerations can be made for the reproduction of Figure 4, which considers a slightly hysteretic junction with $\beta = 1.0$. In the spirit of this challenge I have chosen to show instead [++ the $I - V$ curves of the simulated junction with $\beta = 1.0$ and ++] the related Δi_n vs. α_{rf} curves for the two most significant cases of rf bias, i.e., the sinusoidal drive and the pulsed drive with $\rho = 0.050$ (Fig. 11).

7 Discussion

If I would start today the project from scratch I would make very different choices about the programming languages to use for the simulation and data analysis steps.
Fortran, despite its venerable age, is still an excellent language for scientific programming, but today my language of choice for numerical computations is Python, because of its flexibility, ease of use, and availability of excellent numerical libraries, such as NumPy and SciPy. The main problems with Python are related to the tumultuous development of the language itself and of the thousands of available modules, which can cause incompatibilities even with code developed a few years ago. This problem can be, at least temporarily, be solved by using virtual environments, but a better standardized solution is strongly needed. Another problem with Python is related to its nature of interpreted language. However, the presence of well-documented Fortran and C bindings, can enhance performance of time-critical sections of code.
I would eschew using a new language, as was Visual Basic 1.0 at the time, or as is Apple's Swift today, for a scientific project. It is true that it would have been very hard to foresee in 1993-94 the rapid demise of Microsoft's Visual Basic, nevertheless using a language only when it is stable, runs on a wide array of operating systems and is accepted by a wide community of developers is surely a safe bet. Data analysis today is usually made with R or Python (or to a much lesser extent, Julia or Scala), both open source tools running on all major operation systems, that predictably will be still available in the next 10 or even 20 years.

8 Conclusions

Going back to this old paper has been an exceptionally interesting experience and I must thank the organizers of this challenge for the opportunity offered.
However this is not only a nostalgic attitude. The reproducibility or replicability crisis is a serious issue today [20], when many scientific studies are difficult to reproduce or replicate and true science is challenged daily by cheaters that publish papers with fabricated, falsified, or modified data or results.
Being able to go back and redo what has been done in the past assures that...
[++open source++]
[++ ADD SUPPLEMENTARY INFO??? ++]

References

1. A. Barone and G. Paternò. **Physics and Applications of the Josephson Effect**. Wiley, July 1982.
2. M. Gurvitch, M. A. Washington, and H. A. Huggins. "High quality refractory Josephson tunnel junctions utilizing thin aluminum layers." In: **Applied Physics Letters** 42.5 (Mar. 1983), pp. 472–474.
3. S. P. Benz. "Superconductor normal superconductor junctions for programmable voltage standards." In: **Applied Physics Letters** 67.18 (Oct. 1995), pp. 2714–2716.

4. S. A. Cybart, E. Y. Cho, T. J. Wong, B. H. Wehlin, M. K. Ma, C. Huynh, and R. C. Dynes. "Nano Josephson superconducting tunnel junctions in YBa2Cu3O7−δ directly patterned with a focused helium ion beam." In: **Nature Nanotechnology** 10.7 (July 2015), pp. 598–602.
5. N. De Leo, M. Fretto, V. Lacquaniti, C. Cassiago, L. D'Ortenzi, L. Boarino, and S. Maggi. "Thickness Modulated Niobium Nanoconstrictions by Focused Ion Beam and Anodization." In: **IEEE Transactions on Applied Superconductivity** 26.3 (Apr. 2016), pp. 1–5.
6. K. Likharev and V. Semenov. "RSFQ logic/memory family: a new Josephson-junction technology for sub-terahertz-clock-frequency digital systems." In: **IEEE Transactions on Appiled Superconductivity** 1.1 (Mar. 1991), pp. 3–28.
7. S. Maggi, N. De Leo, V. Lacquaniti, A. Agostino, R. Gonnelli, and P. Verhoeve. "Nb/Al STJ detectors with sub-nA subgap current." In: **Physica C: Superconductivity and its Applications** 435.1-2 (Mar. 2006), pp. 103–106.
8. C. Granata, A. Vettoliere, R. Russo, M. Fretto, N. D. Leo, E. Enrico, and V. Lacquaniti. "Ultra High Sensitive Niobium NanoSQUID by Focused Ion Beam Sculpting." In: **Journal of Superconductivity and Novel Magnetism** 28.2 (Feb. 2015), pp. 585–589.
9. R. Kautz, C. Hamilton, and F. Lloyd. "Series-array Josephson voltage standards." In: **IEEE Transactions on Magnetics** 23.2 (Mar. 1987), pp. 883–890.
10. R. Monaco. "Enhanced ac Josephson effect." In: **Journal of Applied Physics** 68.2 (July 1990), pp. 679–687.
11. D. Andreone, V. Lacquaniti, and S. Maggi. "Experiments on Josephson Junctions Driven by a Bi-Harmonic RF Source." In: **Nonlinear Superconductive Electronics and Josephson Devices**. Boston, MA: Springer US, 1991, pp. 37–43.
12. D. Andreone, V. Lacquaniti, and S. Maggi. "Numerical and Experimental Results on Josephson Junctions Irradiated by a Biharmonic Drive." In: **Superconducting Devices and Their Applications**. 1992, pp. 399–402.
13. S. Maggi. "Step width enhancement in a pulse driven Josephson junction." In: **Journal of Applied Physics** 79.10 (May 1996), pp. 7860–7863.
14. S. Maggi. "Enhanced phase locking in a Josephson junction driven by current pulses." In: **Journal of Low Temperature Physics** 106.3-4 (Feb. 1997), pp. 399–404.
15. R. W. Henry and D. E. Prober. "Electronic analogs of double junction and single junction SQUIDs." In: **Review of Scientific Instruments** 52.6 (June 1981), pp. 902–914.
16. D. E. McCumber. "Effect of ac Impedance on dc Voltage Current Characteristics of Superconductor Weak Link Junctions." In: **Journal of Applied Physics** 39.7 (June 1968), pp. 3113–3118.
17. W. C. Stewart. "Current voltage characteristics of superconducting tunnel junctions." In: **Journal of Applied Physics** 45.1 (Jan. 1974), pp. 452–456.
18. D. G. McDonald, E. G. Johnson, and R. E. Harris. "Modeling Josephson junctions." In: **Physical Review B** 13.3 (Feb. 1976), pp. 1028–1031.
19. A. Boyanski. **FPP - A Fortran Preprocessor**. Tech. rep. Department of Energy, 1992, pp. 1–7.
20. T. Miyakawa. "No raw data, no science: another possible source of the reproducibility crisis." In: **Molecular Brain** 13.1 (Feb. 2020), pp. 1–6.

CAMBIARE ?

FULL PAGE MAIN TEXT HERE

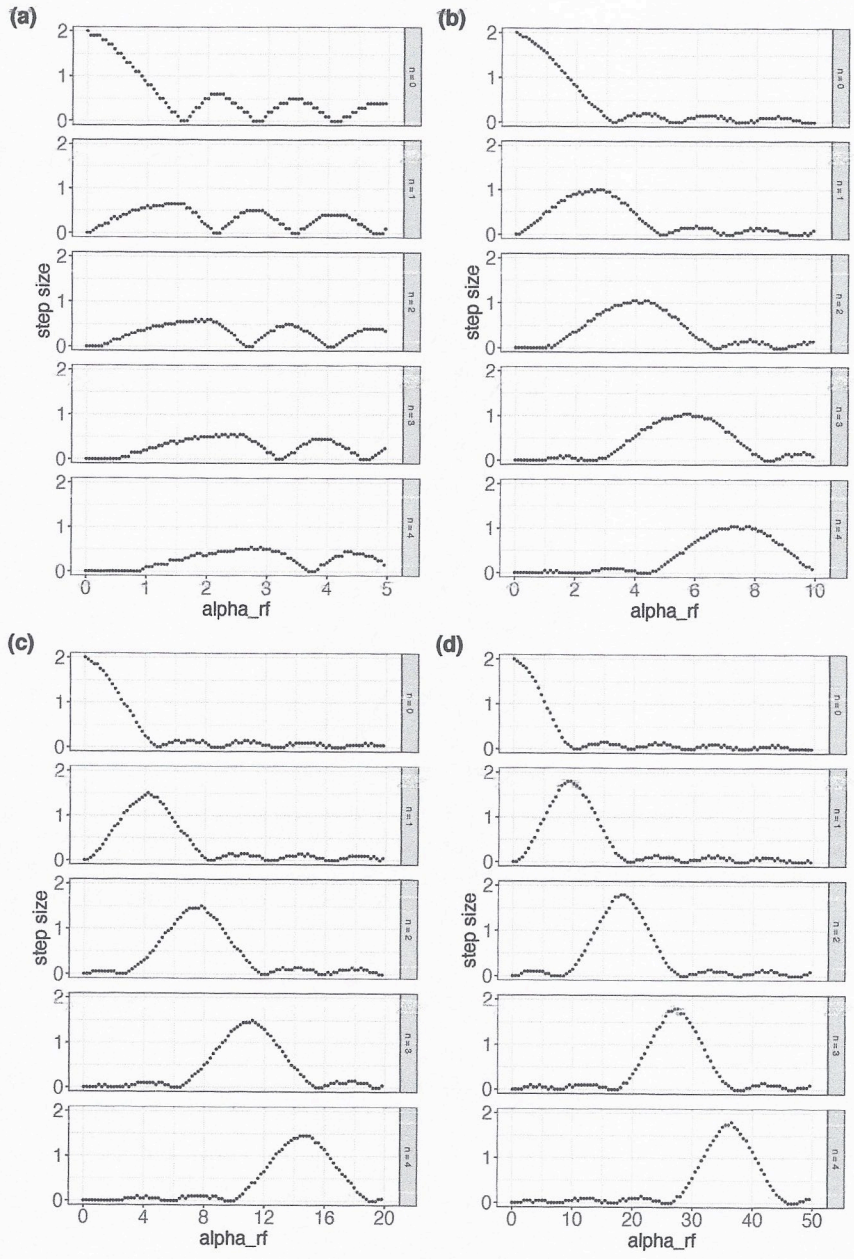

Figure 9. Dependence of the step size Δi_n on the amplitude of α_{rf}, for $n = 0...4$. The $n = 0$ step is the normalized total critical current Δi_0 of the junction. The junction parameters are $\beta = 0.01$ and $\Omega = 0.45$; (a) sinusoidal drive, (b) pulsed drive with $\rho = 0.250$, (c) pulsed drive with $\rho = 0.125$, (d) pulsed drive with $\rho = 0.050$.